МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Создавай уникальные мобильные приложения при помощи последней версии Android SDK!

Android™ 2
Программирование приложений

для планшетных компьютеров и смартфонов

Рето Майер





Android 2. Программирование приложений для планшетных компьютеров и смартфонов

Рето Майер



Москва
2011

Professional Android 2 Application Developmentecond Edition

Reto Meier



Wiley Publishing, Inc.

УДК 004.42
ББК 32.973.26
М 12

Перевод с английского и редакция ООО «Айдиономикс»

Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana. All rights reserved.
This translation published under Licence

Майер Р.

М 12 Android 2 : программирование приложений для планшетных компьютеров и смартфонов : [пер. с англ.] / Рето Майер. — М. : Эксмо, 2011. — 672 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-699-50323-0

Данная книга является наилучшим руководством для программистов, желающих научиться создавать приложения для мобильной платформы Android. Она представляет собой практический курс по написанию программного обеспечения на базе второй версии Android SDK (набора инструментов для разработки программного обеспечения). Это означает, что все теоретические сведения закрепляются максимально приближенными к реальным задачам примерами. Изложение материала предполагает, что читатель владеет основами программирования и базовым уровнем языка Java (второе желательно, но не обязательно).

Информация, которая в ней содержится, будет полезной как для опытных разработчиков (они могут использовать ее как справочник, пропустив первые, элементарные главы), так и для тех, кто делает свои первые шаги в сфере написания мобильных приложений для Android.

УДК 004.42
ББК 32.973.26

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм. Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения ООО «Издательство «Эксмо».

ISBN 978-5-699-50323-0

© ООО «Айдиономикс», перевод на русский язык, 2011
© ООО «Издательство «Эксмо», 2011

Оглавление

Оглавление 5
Об авторе 13
О техническом редакторе 13
Благодарности 13

Введение 14
Для кого предназначена эта книга 15
Какой круг тем охватывает книга 16
Структура книги 17
Что вам понадобится при чтении этой книги 17
Соглашения и условные обозначения 17
Исходный код 18
Список опечаток 19
P2P.WROX.com 19

Глава 1. Здравствуй, Android 21
Небольшая предыстория 22
Чем Android не является 24
Android: открытая платформа для разработки мобильных приложений 25
Встроенные приложения Android 26
Основные характеристики среды разработки для платформы Android 27
Несколько слов об Альянсе открытых мобильных устройств (Open Handset Alliance, OHA) 32

На каких устройствах работает Android 32

Зачем нужно заниматься разработкой ПО для мобильных устройств 33

Для чего нужно заниматься разработкой приложений для Android 34

Знакомство с фреймворком разработчика 37

Резюме 42

Глава 2. Приступаем к работе 44

Разработка приложений для Android 45

Разработка приложений для мобильных устройств 60

Приложение To-Do List 71

Средства разработки для Android 78

Резюме 83

Глава 3. Создание приложений и Активностей 85

Из чего состоят приложения в Android 86

Знакомство с манифестом приложения 88

Использование редактора манифеста 94

Жизненный цикл приложения в Android 96

Приоритеты приложений и состояния процессов 96

Отделение ресурсов от кода программы 98

Знакомство с классом Application в Android 116

Детальный обзор Активностей в Android 119

Резюме 128

Глава 4. Создание пользовательского интерфейса 130

Основы проектирования пользовательского интерфейса в Android 131

Знакомство с Представлениями 132

Знакомство с менеджерами компоновки 135

Создание новых Представлений 138
Ресурсы Drawable 160
Интерфейсы, не зависящие от разрешения и плотности пикселов 169
Создание и использование меню....................176
Резюме 192

Глава 5. Намерения, Широковещательные приемники, Адаптеры и Интернет 194
Знакомство с Намерениями....................195
Знакомство с Ожидающими намерениями 225
Знакомство с Адаптерами....................225
Ресурсы Интернета 233
Знакомство с диалоговыми окнами....................236
Создание приложения Earthquake Viewer....................241
Резюме 251

Глава 6. Файлы, сохранение состояния и настройки 252
Сохранение простых данных приложения 253
Создание и сохранение настроек....................254
Получение Общих настроек....................254
Создание Активности для настроек приложения Earthquake Viewer....................255
Знакомство с Активностью настроек и фреймворком для их создания 263
Создание стандартной Активности настроек для приложения Earthquake Viewer....................269
Сохранение состояния Активности 271
Сохранение и загрузка файлов 275
Включение статических файлов в качестве ресурсов....................276
Инструменты для управления файлами....................276
Резюме 277

Глава 7. Базы данных и Источники данных 278
Введение в базы данных на платформе Android 278
Введение в SQLite 280
Курсоры и класс ContentValues 280
Работа с базами данных SQLite 281
Создание нового Источника данных 295
Использование Источников данных 299
Создание и использование Источника данных для приложения Earthquake 302
Стандартные Источники данных в Android 310
Резюме 317

Глава 8. Карты, геокодирование и геолокационные сервисы 319
Использование геолокационных сервисов 320
Настройка эмулятора для тестирования геолокационных сервисов 320
Изменение местоположения в эмуляторе с помощью LocationProvider 321
Выбор Источника данных для получения местоположения 322
Поиск вашего местоположения 324
Использование оповещений о близости нахождения 331
Использование геокодировщика 333
Создание Активностей, основанных на MapView 338
Добавление картографических возможностей в проект Earthquake 360
Резюме 365

Глава 9. Работа в фоновом режиме 367
Знакомство с Сервисами 368
Использование фоновых потоков 384
Вывод уведомлений типа Toast 390
Знакомство с уведомлениями 394

Использование Сигнализации 408
Резюме 415

Глава 10. Домашний экран в Android 417
Знакомство с виджетами на домашнем экране 418
Создание виджетов 419
Создание виджета для приложения Earthquake 433
Знакомство с Живыми каталогами 439
Виджет быстрого поиска и добавление поисковых возможностей в свое приложение 446
Создание Живых обоев 455
Резюме 458

Глава 11. Аудио, видео и камера 459
Проигрывание аудио и видео 460
Запись аудио- и видеоданных 468
Использование камеры и создание снимков 473
Добавление новых мультимедийных данных в MediaStore 481
Работа с несжатым звуком 482
Распознавание речи 486
Резюме 488

Глава 12. Телефония и SMS 489
Телефония 490
Знакомство с SMS и MMS 500
Резюме 527

Глава 13. Bluetooth, Wi-Fi, Сеть 528
Использование Bluetooth 528
Управление сетевыми соединениями 555
Управление подключением к сети Wi-Fi 559
Резюме 563

Глава 14. Датчики ... 565
Использование датчиков и объекта sensormanager ... 566
Интерпретация данных, полученных с помощью датчиков ... 570
Использование компаса, акселерометра и датчика ориентации ... 571
Управление вибрацией устройства ... 587
Резюме ... 588

Глава 15. Продвинутое программирование под Android ... 589
Paranoid Android ... 590
Преобразование текста в речь на платформе Android ... 594
Использование AIDL при межпроцессном взаимодействии Сервисов ... 597
Использование интернет-сервисов ... 602
Создание насыщенных пользовательских интерфейсов ... 603
Резюме ... 646

Алфавитный указатель ... 648



Посвящается Кристи

Об авторе

Рето Майер родом из города Перт, Западная Австралия. Сейчас проживает в Лондоне.

Работает в компании Google, помогая разработчикам создавать лучшие приложения для платформы Android. Рето — опытный программист: за его плечами более 10 лет проектирования приложений с графическим интерфейсом. Прежде чем перейти в Google, он сменил несколько профессий, в том числе трудился в финансовой и нефтегазовой сфере.

Рето принимает участие в проекте Android с момента его первого выпуска в 2007 году. В свободное время совершенствует различные платформы для разработки, применяя весь арсенал инструментов компании Google.

У Рето есть блог — http://blog.radioactiveyak.com, а на странице http://www.twitter.com/retomeier вы можете подписаться на его твиттер.

О техническом редакторе

Милан Нарендра Шах получил степень бакалавра компьютерных наук в Университете Саутгемптона. Работает инженером-программистом более 7 лет, знает языки C#, C/C++, Java. Вместе с женой проживает в графстве Бакингемшир, Великобритания.

Благодарности

Прежде всего я бы хотел поблагодарить Кристи. Все, что я делаю, становится возможным благодаря твоей поддержке. Ты помогла сделать эту книгу настолько хорошей, насколько это вообще возможно. Без тебя она бы никогда не была написана.

Я бы хотел выразить большую признательность команде Google, особенно инженерам, развивающим платформу Android, и коллегам из отдела по связям с разработчиками. Темпы, с которыми Android рос и совершенствовался в последние годы, воистину феноменальны.

Хочу также сказать спасибо Скотту Майерсу за то, что дал мне возможность обновить книгу, а также Биллу Бриджесу, Милану Шаху, Сэйди Клейнману и команде издательства Wrox за то, что помогли ее создать.

Отдельная благодарность сообществу разработчиков для Android. Ваш упорный труд и невероятные приложения составили основу успеха этой платформы.

ВВЕДЕНИЕ

Сегодня перед разработчиками для мобильных платформ открываются потрясающие возможности. Мобильные телефоны еще никогда не были такими популярными, а мощные смартфоны теперь общедоступны. В стильных и многофункциональных устройствах такие аппаратные возможности, как GPS, акселерометры и сенсорные экраны сочетаются с фиксированными и адекватными тарифами на передачу данных по Сети, благодаря чему вы получаете доступ к привлекательной платформе и имеете возможность создавать для нее инновационные мобильные приложения.

Сейчас у покупателей широкий выбор телефонов под управлением Android. Кроме устройств с экранами, имеющими разрешение QVGA, существуют мощные модели с поддержкой WVGA: Motorola Droid и Google Nexus One. В итоге в выигрыше не только пользователи, но и разработчики. В отличие от большинства мобильных систем, закрывающих и ограничивающих разработку и развертывание сторонних приложений, Android предлагает альтернативу: позволяет писать приложения, использующие всю мощь современного аппаратного обеспечения, и распространять их в условиях открытого рынка.

В результате такой политики вместе с ростом продаж мобильных телефонов бешеными темпами растет и заинтересованность разработчиков в платформе Android. В 2009 и в начале 2010 года HTC, Motorola, LG, Samsung и Sony Ericsson выпустили свыше 20 миллионов аппаратов под управлением Android. Сейчас устройства на этой платформе доступны более чем в 26 странах, а число операторов мобильной связи, которые их поддерживают, свыше 30. В Соединенных Штатах аппараты на базе Android распространяются в сетях четырех крупнейших провайдеров: T-Mobile, Verizon, AT&T и Sprint. Кроме того, вы можете купить разблокированную модель Google Nexus One прямо на сайте Google по адресу http://www.google.com/phone.

Android, базирующийся на свободном фреймворке с мощными библиотеками в составе SDK и придерживающийся принципов открытости, предоставил возможность создавать собственные мобильные приложения тысячам разработчиков, у которых ранее не было доступа к соответствующим инструментам. Благодаря этой платформе опытные программисты могут использовать новые уникальные возможности для созданиях инновационных продуктов или улучшения существующих.

Задействовав Android Market для дистрибуции платных и бесплатных приложений на любые устройства, совместимые с Android, разработчики могут использовать все преимущества открытого рынка, избегая при этом дополнительных проверок.

Эта книга — практическое руководство по созданию мобильных приложений с учетом второй версии Android SDK (набора инструментов для разработки программного обеспечения). Вы познакомитесь с демонстрационными проектами, каждый из которых содержит новые функции и механизмы, предоставляемые платформой Android. Книга охватывает весь основной потенциал системы, раскрывает дополнительные возможности на кратких примерах.

Философия Google — выпускать продукты как можно раньше и чаще. С момента выхода первой версии Android в октябре 2008 года увидело свет семь крупных обновлений платформы и SDK. С учетом столь быстрого темпа разработки изменения и улучшения в программные библиотеки вносятся с завидной регулярностью. И хотя инженеры из команды Android прикладывают значительные усилия, чтобы обеспечить обратную совместимость, выход новых версий, вероятно, повлияет на актуальность информации, представленной в этой книге.

Тем не менее, комментарии и примеры этого издания дадут основные знания, необходимые для написания мощных мобильных приложений с помощью текущей версии SDK. Ну, а на базе изученного материала вы сможете быстро приспосабливаться к будущим изменениям.

Для кого предназначена эта книга

Эта книга будет полезна всем, кто заинтересован в создании приложений для мобильной платформы Android с использованием SDK: и опытным разработчикам, и тем, кто только делает первые шаги в написании мобильных приложений для Android.

Опыт использования мобильных телефонов (в частности, под управлением Android) желателен, но необязателен, равно как и практика разработки для мобильных устройств. Предполагается, что у вас есть какое-то представление о написании программного обеспечения и о базовых принципах разработки. Пригодится и знание языка Java, хотя это также лишь рекомендательно.

В главах 1 и 2 вы познакомитесь с разработкой для мобильных платформ и получите инструкции, как начать работать с системой Android. Нет никакой необходимости читать главы в том порядке, в котором они идут, хотя понимание основных компонентов, описанных в главах 3–7, также играет важную роль при изучении последующего материала. Главы 8–15 рассказывают про дополнительные и продвинутые функции, вы можете читать их в любой последовательности в зависимости от интересов и потребностей.

Какой круг тем охватывает книга

В главе 1 вы познакомитесь с платформой Android, узнаете, что она из себя представляет и как соотносится с уже устоявшимся процессом разработки для мобильных систем. Затем более подробно рассмотрены возможности, которые предлагает эта платформа для написания мобильных приложений.

Глава 2 предлагает рекомендации по разработке для мобильных платформ. В ней объясняется, как загрузить Android SDK и начать создавать программы. Вы также познакомитесь с инструментами для программирования, доступными в Android, и научитесь проектировать новые приложения с нуля.

В главах 3–7 вы углубитесь в изучение основных программных компонентов. Начав со знакомства с жизненным циклом приложений в Android и их структурой, перейдете к изучению программного манифеста и внешних ресурсов. После этого уделите внимание Активностям (Activity), их жизненным циклам и режимам работы.

Вы узнаете, как создавать пользовательский интерфейс с помощью разметки и Представлений (Views), а затем познакомитесь с механизмом Намерений (Intents), который используется для выполнения различных действий и обмена сообщениями между программными компонентами. После знакомства с интернет-ресурсами пристальное внимание уделено хранению, поиску и совместному использованию информации. Вы начнете с механизма сохранения настроек, после перейдете к работе с файлами и базами данных. В завершение на примере увидите, как обмениваться информацией с помощью Источников данных (Content Providers).

В главах 8–14 рассматриваются более сложные темы: картографические и геолокационные функции, Сервисы (Services), фоновые потоки и механизм уведомлений. Вы узнаете, как с помощью виджетов, Живых каталогов (Live Folders), Живых обоев (Live Wallpaper)и панели быстрого поиска приложения могут взаимодействовать с пользователем прямо на домашнем экране. Познакомившись с процессом проигрывания и записи мультимедийных данных, а также научившись работать с камерой, рассмотрите коммуникационные возможности Android.

Программные интерфейсы, обеспечивающие функции телефонии, рассмотрены наряду с API для отправки и получения SMS-сообщений. Уделено внимание и управлению Bluetooth и сетью (как Wi-Fi, так и мобильными подключениями).

В главе 14 вы изучите API для работы с датчиками и научитесь использовать компас, акселерометры и другие аппаратные сенсоры, с помощью которых приложение сможет реагировать на изменения внешней среды.

Глава 15 предлагает продвинутые разработки: безопасность, IPC, нестандартные методы работы с графикой и взаимодействие пользователя с аппаратным обеспечением.

Структура книги

Книга состоит из глав, упорядоченных таким образом, чтобы помочь читателям с разным опытом и навыками научиться писать сложные приложения для Android.

Необязательно изучать материал именно в той последовательности, в которой он изложен, но некоторые демонстрационные проекты разрабатываются на протяжении нескольких глав, а на каждом следующем этапе дополняются новыми функциями и улучшениями.

Программисты с опытом работы со средой Android SDK могут пропустить первые две главы, в которых происходит знакомство с особенностями мобильных платформ и даются инструкции по созданию среды разработки, и сразу приступить к главам с 3-й по 7-ю. Они охватывают фундаментальные принципы разработки для Android, поэтому важно иметь глубокое понимание тех концепций, которые в них описываются. Разобравшись с основами, можете переходить к оставшимся главам, где речь идет о картографии, геолокационных сервисах, фоновых приложениях, а также о более продвинутых аспектах разработки, таких как взаимодействие с аппаратным обеспечением и сетями.

Что вам понадобится при чтении этой книги

Для использования демонстрационных проектов этой книги нужно создать среду разработки, загрузив Android SDK, JDK (Java Development Kit) и сопутствующие инструменты. При желании можете загрузить и установить среду Eclipse и дополнение ADT (Android Developer Tool), которые помогут упростить процесс разработки.

Android SDK поддерживает ОС Windows, MacOS и Linux, его можно загрузить с официального сайта.

Чтобы применить информацию из этой книги или разрабатывать приложения, не нужно иметь устройства под управлением Android.

ПРИМЕЧАНИЕ

В главе 2 все требования будут рассмотрены более подробно. Кроме того вы узнаете, где можно загрузить и как установить каждый из упоминавшихся ранее компонентов.

Соглашения и условные обозначения

Чтобы вы смогли извлечь для себя максимальную пользу от данного текста, не теряя нить повествования, я использовал некоторые условные обозначения.

ВРЕЗКИ

Примечания, советы, подсказки, приемы и ремарки, относящиеся к основным темам, оформляются, как этот текст.

Что касается текстовых стилей.

- Адреса URL я выделяю так: Wrox.com.
- Код я выделяю так: `persistence.properties`.
- Чтобы улучшить читаемость текста, названия компонентов начинаются с прописных букв и оформлены так: Источник данных[1].
- Я выделяю код двумя различными способами:

```
Я использую моноширинный шрифт без выделений для большинства фрагментов кода.
```

```
Я использую полужирный шрифт, чтобы обратить ваше внимание на код, который играет важную роль в данном контексте.
```

- В некоторых фрагментах кода вы можете увидеть строки, обозначенные следующим образом:

```
[ ... ранее написанный код ... ]
```

  или

```
[ ... реализуйте здесь что-нибудь ... ]
```

  Такие инструкции говорят о том, что всю строку (включая квадратные скобки) нужно заменить на реальный код, который содержится в предыдущем фрагменте (как в первом случае) либо должен быть реализован вами самими, но позже.

- Чтобы примеры кода были достаточно лаконичными, я не всегда включаю в них все необходимые операторы import. Демонстрационные проекты, о которых речь пойдет чуть ниже, содержат полноценные исходники, в том числе импорт всех требуемых пакетов.

Исходный код

Работая с примерами из этой книги, вы можете либо набирать весь код вручную, либо воспользоваться файлами, которые можно загрузить с wrox.com[2]. Зайдя на сайт, найдите название книги (с помощью поисковой

[1] При первом упоминании подобных компонентов дается их англоязычное название. — *Примеч. ред.*

[2] Все указанные в книге сайты англоязычные. Издательство не несет ответственности за их содержимое и напоминает, что со времени написания книги сайты могли измениться или вовсе исчезнуть. — *Примеч. ред.*

строки или через один из списков), пройдите к ее описанию и щелкните на ссылке Download Code, чтобы получить все исходники.

ПРИМЕЧАНИЯ

Поскольку названия многих книг совпадают, лучше всего искать по коду ISBN, а именно: 978-0-470-56552-0.

Загрузив код, распакуйте его с помощью любимого архиватора. Вы также можете пройти по адресу www.wrox.com/dynamic/books/download.aspx и ознакомиться с исходниками, доступными для этой и всех остальных книг, опубликованных издательством Wrox.

Список опечаток

Мы делаем все возможное, чтобы не допускать ошибок в тексте или коде. Однако от этого никто не застрахован. Если вы обнаружите неточности в какой-либо из наших книг, будь то орфографическая ошибка или неправильный участок кода, мы будем признательны, если вы нам о них сообщите. Присылая найденные опечатки, вы можете сэкономить другим читателям время и при этом помочь нам обеспечить еще более высокое качество материала.

Чтобы попасть на страницу с опечатками для этой книги, зайдите на сайт wrox.com и с помощью поисковой строки или одного из списков найдите соответствующее название, затем пройдите по ссылке Book Errata. Там вы сможете просмотреть список всех неточностей, которые были обнаружены в этой книге и опубликованы редакторами издательства Wrox. Полный список изданий размещен по адресу wrox.com/misc-pages/booklist.shtml и содержит ссылки на опечатки в каждой книге.

Если вы не хотите пользоваться разделом Book Errata, можете пройти на страницу wrox.com/contact/techsupport.shtml и отправить найденные вами ошибки, заполнив соответствующую форму. Мы проверим полученную информацию и при необходимости опубликуем сообщение в разделе с ошибками, сделав исправления для следующих изданий книги.

p2p.wrox.com

Чтобы подискутировать с автором и с другими читателями, присоединяйтесь к конференции p2p.wrox.com. Это система интернет-форумов, где можно публиковать сообщения, относящиеся к книгам издательства Wrox и связанным с ними технологиям. Там вы сможете пообщаться с другими читателями и пользователями, подписаться на интересные для вас темы и получать новые сообщения по электронной почте.

На сайте http://p2p.wrox.com разворачиваются тематические обсуждения, которые пригодятся не только при чтении книги, но и при разработке собственных приложений. Чтобы присоединиться к конференции, достаточно выполнить следующие действия:

1) пройдите на сайт p2p.wrox.com и щелкните на ссылке Register Now;
2) ознакомьтесь с условиями использования форума и нажмите кнопку Register;
3) введите необходимые данные, а также любую дополнительную информацию, которую хотите указать, и щелкните на Complete Registration;
4) получите электронное письмо с информацией, как подтвердить свою учетную запись и завершить процесс регистрации.

ПРИМЕЧАНИЕ

Чтобы просматривать конференцию, регистрироваться необязательно. Это нужно только в том случае, если вы хотите публиковать собственные сообщения.

Присоединившись к конференции, вы можете писать сообщения и отвечать другим пользователям, просматривать форум в любое время. Если вы хотите получать по электронной почте новые сообщения из определенного раздела, щелкните на значке Subscribe to This Forum рядом с его названием.

За более детальной информацией о том, как использовать Wrox P2P, обратитесь к списку часто задаваемых вопросов (FAQ). Помимо множества интересных сведений о P2P и книгах, изданных компанией Wrox, вы сможете узнать, например, на каком программном обеспечении базируется система форумов. Для этого пройдите по ссылке FAQ на любой странице конференции.

Глава 1
ЗДРАВСТВУЙ, ANDROID

Содержание главы

- Основы разработки мобильных приложений.
- Чем является и чем не является Android.
- Основные характеристики среды разработки для платформы Android.
- На каких устройствах работает Android.
- Зачем нужны программы для мобильных устройств и платформы Android.
- Описание среды разработки и фреймворка Android.

Независимо от того, опытный вы инженер в области мобильных приложений, разработчик компьютерных программ, веб-программист или любитель, Android предоставляет отличную возможность по написанию инновационных приложений для мобильных устройств.

Несмотря на название, Android не предназначен для создания несокрушимой армии хладнокровных роботов-солдат с целью очистить Землю от гнета человечества. Android представляет собой набор программ с открытым исходным кодом, который включает операционную систему, подпрограммное обеспечение и ключевые мобильные приложения вместе с библиотеками API, предназначенными для написания новых программ, определяющих визуальное представление и функционал мобильных устройств.

Самые разнообразные компактные стильные мобильные устройства со временем снабжались такими мощными инструментами, как камера, медиаплеер, навигатор, сенсорный дисплей. С внедрением новых технологий мобильный телефон превратился в нечто большее, чем просто устройство для звонков. При этом программная платформа и среда разработки отставали от бешеного темпа развития технологий.

До сегодняшнего дня мобильные телефоны работали под управлением закрытых платформ, построенных на основе сильно фрагментированных запатентованных операционных систем, для чего требовались запатентованные инструменты разработки. Сами же телефоны функционировали с оригинальным программным обеспечением гораздо лучше, чем со сторонним.

Это создавало искусственные препятствия для программистов, которые рассчитывали на использование более мощного аппаратного обеспечения мобильных устройств.

В случае с Android встроенное ПО написано на том же самом API, что и программы сторонних разработчиков, при этом время для исполнения и тех, и других одинаково. Данное API позволяет получить доступ к сенсорному управлению, навигационным сервисам, фоновым и картографическим процессам, реляционным базам данных, двух- и трехмерной графике, к функциям видеозаписи, межпрограммного взаимодействия.

В данной книге вы познакомитесь с функционалом API, что позволит разрабатывать приложения для платформы Android. В настоящей главе мы рассмотрим общие принципы создания мобильных программ, а также изучим основные возможности среды разработки.

В распоряжении разработчика приложений для Android достаточно мощное API и качественная справочная документация. Он может стать членом огромного сообщества, ему не нужно тратиться на программное обеспечение или рекламу своего продукта. С ростом популярности мобильных устройств открываются великолепные перспективы разрабатывать инновационные мобильные приложения, причем с любым опытом программирования.

Небольшая предыстория

Задолго до сетей Twitter и Facebook, когда Google был всего лишь идеей в головах его создателей, а по Земле бродили динозавры, мобильный телефон представлял собой переносное устройство связи, достаточно компактное, чтобы поместиться в чемодане, а его батарейки хватало на несколько часов работы. Тем не менее он давал достаточно свободы, чтобы совершать звонки без физического подключения к телефонной линии.

Теперь в нашу жизнь прочно вошли компактные, стильные и мощные мобильные телефоны, став незаменимой вещью. Благодаря развитию электроники телефоны стали меньше и функциональнее, а число входящих в них периферийных устройств выросло.

После добавления камер и плееров появились телефоны с GPS-навигаторами, акселерометрами и сенсорными дисплеями. Казалось, что рост аппаратных возможностей подготовил благодатную почву для разработчиков программного обеспечения, однако на деле приложения для мобильных телефонов в своем развитии сильно отстали от их аппаратной части.

Не такое далекое прошлое

Исторически сложилось, что программисты, работающие на низкоуровневом C или C++, должны были разбираться в специфике устройств, для

которых они создавали программное обеспечение (будь то одно устройство или целая их серия любого производителя). С развитием аппаратных возможностей и расширением доступа к мобильному Интернету данный подход стал неактуальным.

В последнее время созданы платформы, например Symbian, обеспечивающие разработчикам доступ к более широкой аудитории потребителей ПО. Эти системы подталкивали программистов к написанию большого количества приложений, которые эффективно использовали имеющиеся аппаратные средства.

Созданные платформы открыли доступ к некоторым устройствам аппаратной части, но для этого требовалось писать полноценный код C/C++ с использованием запатентованного «тяжелого» API, работа с которым была чрезвычайно сложной. Дополнительные трудности возникали при функционировании приложений на устройствах с различным исполнением аппаратной начинки, в особенности это касалось устройств с GPS-навигаторами.

За последние годы настоящим прорывом в развитии ПО для мобильных телефонов стало изобретение мидлетов для платформы Java. Мидлеты исполняются на виртуальной Java-машине, что позволяет абстрагироваться от архитектуры того или иного устройства и создавать приложения, работающие на любом из них, которое поддерживает Java. К сожалению, за такое удобство приходится платить ограниченными возможностями доступа к аппаратной части.

В программировании ПО для мобильных устройств совершенно нормально, что приложения сторонних разработчиков получают доступ к аппаратной начинке, при этом им выделяются такие же права на исполнение, как и встроенному ПО, которое было разработано производителем телефона. К сожалению, в случае с мидлетами обе эти возможности ограничены.

Изобретение мидлетов для платформы Java привлекло большое количество разработчиков, однако отсутствие низкоуровневого доступа к аппаратной части и ограниченные возможности исполнения кода означали, что большинство мобильных приложений представляют собой стандартные desktop-приложения или специальные веб-сайты, приспособленные к отображению на небольших экранах. Таким образом, мобильные приложения никак не использовали преимущества и возможности портативных платформ.

Будущее

Android — одна из операционных систем нового поколения, созданных для работы с аппаратным обеспечением современных мобильных устройств. На сегодняшний день Windows Mobile, Apple iPhone и Palm Pre предлагают достаточно мощные и более простые в использовании среды разработки

мобильных приложений. Однако в отличие от Android это запатентованные операционные системы, в которых в определенных случаях приоритет отдается встроенному ПО, а не приложениям сторонних программистов. Кроме того, эти операционные системы ограничивают возможности взаимодействия приложений с данными телефона, а также ограничивают или контролируют процесс распространения сторонних приложений, созданных для данных платформ.

Android дает новые возможности для мобильных приложений, предлагая открытую среду разработки, построенную на открытом ядре Linux. У всех приложений есть доступ к аппаратным средствам устройства, для чего используются специальные серии API-библиотек. Кроме того, здесь включена полная и контролируемая поддержка взаимодействия приложений.

На платформе Android все программы имеют одинаковый статус. Сторонние приложения написаны на том же API, что и встроенное ПО, при этом во всех программах одинаковое время исполнения. Пользователи могут удалять или заменять встроенные ПО на альтернативные сторонние разработки, будь то номеронабиратель или Рабочий стол.

Чем Android не является

Неудивительно, что внедрение революционной технологии вызвало некоторое недопонимание, чем на самом деле является Android. Можно уверенно сказать, чем он не является.

- **Реализацией платформы Java ME.** Приложения для Android написаны на языке Java, однако они не исполняются на виртуальной машине Java ME, соответственно, скомпилированные для Java классы и исполняемые файлы не будут работать на платформе Android.
- **Частью инициативы Linux Phone Standards Forum (LiPS) (Форума стандартов мобильного Linux) или Open Mobile Alliance (OMA) (Альянса открытых мобильных архитектур).** Android работает на Linux-ядре с открытым исходным кодом, но при всей схожести целей весь программный стек платформы Android призван решать более перспективные задачи по сравнению с инициативами упомянутых организаций по определению стандартов.
- **Стандартным уровнем приложений (например, UiQ или S60).** Хотя Android имеет уровень приложений, он также охватывает весь программный стек, начиная с самого нижнего уровня — операционной системы — и заканчивая уровнем библиотек API и самих приложений.
- **Видом мобильных телефонов.** Android включает референс-дизайн для производителей мобильных телефонов, однако в природе не существует телефона марки Android. Как раз наоборот: Android создали для использования на самых разнообразных мобильных устройствах.

- **Ответом Google на iPhone.** iPhone представляет собой запатентованную программную и аппаратную платформу компании Apple. В то же время Android — это набор программ с открытым исходным кодом, разработанный и поддерживаемый Альянсом открытых мобильных архитектур. Android предназначен для работы на любых мобильных устройствах, которые отвечают определенным требованиям. Хотя Google выпустил свой первый, ориентированный на конечного потребителя коммуникатор Nexus 1, его можно назвать всего лишь одним из аппаратных решений, работающим на платформе Android.

Android: открытая платформа для разработки мобильных приложений

Вот как описывает Android Энди Рубин из Google:

*Первая действительно открытая и всеобъемлющая платформа для мобильных устройств и любого программного обеспечения, предназначенного для работы на мобильном телефоне, при этом без патентных ограничений, которые сдерживали развитие портативных устройств. (*http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html*).*

Упрощенно Android можно представить как комбинацию трех компонентов:

- свободной операционной системы с открытыми исходными кодами;
- среды разработки с открытыми исходными кодами для создания мобильных приложений;
- устройств, по большей части мобильных телефонов, на которых установлена операционная система Android вместе с разработанными для нее приложениями.

Android включает несколько необходимых и взаимозависимых элементов:

- референс-дизайн аппаратного обеспечения с перечнем требований к мобильным устройствам, чтобы гарантировать совместимость с ПО;
- ядро операционной системы Linux, которое предоставляет низкоуровневый интерфейс для управления аппаратным обеспечением, памятью и процессами, оптимизированными для работы на мобильных устройствах;
- библиотеки с открытыми исходными кодами, предназначенными для разработки приложений SQLite, WebKit, OpenGL и медиаменеджер;
- среду исполнения для приложений, включающую виртуальную машину Dalvik и библиотеки ядра, которые отвечают за функционал

Android; среда исполнения отличается небольшим размером, что позволяет эффективно использовать ее на мобильных устройствах;

- набор программных компонентов, обеспечивающих доступ к системным службам на уровне приложений; среди них менеджер окон и менеджер местоположения, контент-провайдеры, возможности работы с телефонией и сенсорным дисплеем;
- набор компонентов пользовательского интерфейса для размещения и запуска приложений;
- предустановленные приложения, поставляемые в общем программном наборе;
- комплект программ для разработки приложений, включающий инструменты, плагины и справочную документацию.

Особо стоит подчеркнуть, что открытая архитектура Android позволяет исправлять любые ошибки в пользовательском интерфейсе или дизайне встроенных приложений путем написания расширений или замещений ошибок. Android предоставляет возможность создавать собственные интерфейсы для мобильных телефонов, а также приложения с функционалом и дизайном, максимально отвечающими вашим потребностям.

Встроенные приложения Android

Телефоны с системой Android снабжены набором предустановленных программ, разработанных в рамках проекта Android Open Source Project (AOSP) (Проект открытых исходных кодов для Android).

Перечислим основные из них:

- e-mail-клиент;
- приложение для работы с SMS;
- полный набор инструментов для управления личными данными пользователя, включая календарь и адресную книгу;
- браузер на базе WebKit;
- музыкальный плеер и фотогалерея;
- калькулятор;
- Рабочий стол;
- будильник.

Во многих случаях Android включает также следующее лицензионное ПО от Google:

- приложение Android Market для загрузки сторонних программ, разработанных для платформы Android;

- полноценное приложение Google Maps, включающее функции StreetView («Просмотр улиц»), Driving Directions («Показ проезда»), маршрутизируемую навигацию, спутниковую карту и информацию о пробках;
- программу для работы с почтой Gmail;
- программу для обмена мгновенными сообщениями Google Talk;
- видеоплеер для работы с сервисом YouTube.

Данные, к которым имеют доступ многие из этих приложений, например адресная книга, открыты и для программ сторонних разработчиков. Кроме этого приложения могут обрабатывать такие события, как входящий звонок или получение SMS.

Внешний вид программ, которые установлены на новых телефонах под управлением Android, может сильно варьироваться в зависимости от производителя аппаратного обеспечения и/или оператора, дистрибьютора.

Открытый характер платформы Android означает, что операторы или производители комплектного оборудования (OEM) могут менять пользовательский интерфейс и набор программ на любом устройстве под управлением Android. Некоторые производители разработали свои собственные интерфейсы на базе Android, например Sense от HTC, MotoBlur от Motorola и пользовательский интерфейс от Sony Ericsson.

Важно отметить, что для всех совместимых устройств платформа и среда разработки остаются неизменными независимо от производителя или оператора. Пользовательский интерфейс может меняться, однако программы будут работать абсолютно одинаково на всех совместимых с Android устройствах.

Основные характеристики среды разработки для платформы Android

Главным сокровищем Android как среды разработки стал ее API.

Android как нейтральная к приложениям платформа предоставляет возможность создавать программы, которые станут такой же неотъемлемой частью телефона, как и компоненты, поставляемые в комплекте.

Следующий список иллюстрирует основные характеристики Android:

- отсутствие расходов на использование лицензии, распространение и разработку, а также каких-либо механизмов сертификации готовых программных продуктов;
- доступ к Wi-Fi-устройству;
- в сетях GSM, EDGE и 3G, предназначенных для телефонии и передачи данных, можно звонить или принимать звонки и SMS, отправлять и получать данные;

- комплексный API для работы с навигационными службами, например GPS;
- полный контроль над мультимедийными устройствами, включая проигрывание или запись информации с камеры и микрофона;
- API для работы с сенсорными устройствами, например акселерометром и компасом;
- библиотеки для работы с Bluetooth с возможностью передачи данных по протоколу p2p;
- передача IPC-сообщений;
- хранилища для общих данных;
- фоновые приложения и процессы;
- виджеты для Рабочего стола, Живые каталоги (Live Folders) и Живые обои (Live Wallpaper);
- возможность интеграции результатов поиска приложения в системный поиск;
- встроенный браузер на базе WebKit с открытыми исходными кодами и поддержкой HTML5;
- полная поддержка приложений, которые используют функционал работы с картами в своем пользовательском интерфейсе;
- оптимизированная под мобильные устройства графическая система с аппаратным ускорением, включающая библиотеку для работы с векторной 2D-графикой и поддержку трехмерной графики с использованием OpenGL ES 2.0;
- мультимедийные библиотеки для проигрывания и записи аудио-, видеофайлов или изображений;
- локализация с помощью инструментов для работы с динамическими ресурсами;
- набор программных компонентов для повторного использования компонентов и замещения встроенных приложений.

Работа с аппаратными ресурсами, включая камеру, GPS-навигатор и акселерометр

В состав Android входят библиотеки API, которые упрощают разработку программ, использующих аппаратные ресурсы устройства. Это значит, что не нужно каждый раз создавать специальные версии программы для разных устройств. Вы можете создать приложение на платформе Android, которое будет работать на любом совместимом с ней устройстве.

Среда разработки для Android включает API для работы с навигационными устройствами (в частности, с GPS-навигатором), камерой, звуковой системой, сетевыми соединениями, Wi-Fi, Bluetooth, акселерометрами, сенсорным дисплеем и системой управления питанием. Более подробная информация о возможностях API для работы с аппаратным обеспечением Android в главах 11–14.

Встроенные службы Google Maps, Geocoding (геокодирование) и сервисы навигации

Android поддерживает работу с картами, а значит можно создавать навигационные приложения, которые будут эффективно использовать мобильные преимущества устройств под управлением Android. Данная платформа позволяет программам включать сервис Google Maps в интерфейс и обеспечивает полный доступ к картам, которыми можно управлять программно и при необходимости снабжать комментариями, используя богатые возможности графической библиотеки Android.

Навигационные сервисы платформы работают с GPS и технологией определения положения по базовым станциям сетей GSM от Google, с помощью которых устанавливается текущее местонахождение устройства. Данные сервисы позволяют абстрагироваться от особенностей той или иной технологии, при этом вы задаете минимальный набор настроек (например, точность или стоимость) и выбираете нужную технологию. Кроме этого платформа гарантирует, что ваши навигационные программы будут работать независимо от того, какую из технологий поддерживает то или иное устройство.

Для соединения карт с навигационными сервисами в состав Android включен API для прямого и обратного геокодирования, который позволяет находить на карте координаты по заданному адресу или определять адрес для определенной позиции на карте.

В главе 8 вы научитесь работать с картами, функцией геокодирования и навигационными сервисами.

Фоновые службы

На платформе Android можно создавать приложения и службы, которые работают в фоновом режиме.

Современные мобильные устройства, как правило, мультизадачные. Однако из-за небольших размеров экранов вы можете видеть только одно интерактивное приложение. Платформы, которые не поддерживают фоновую работу приложений, ограничивают время жизни программ, которые не требуются вам постоянно.

Фоновые службы позволяют создавать невидимые компоненты приложений, которые в автоматическом режиме выполняют какие-либо операции без прямого участия пользователя. Благодаря фоновым процессам можно работать с событиями или регулярно производить обновления. Кроме того, они идеально подходят для мониторинга биржевых сводок и результатов игр, отображения навигационных сообщений или фильтрации входящих звонков и сообщений.

Более подробно вы познакомитесь с фоновыми службами в главе 9.

Использование баз данных SQLite для хранения и извлечения информации

Для устройств, размеры которых не позволяют использовать большие объемы памяти, как никогда актуально быстрое и эффективное сохранение и извлечение информации.

У каждого приложения, работающего на платформе Android, есть доступ к легковесной реляционной базе данных SQLite. Ваша программа может использовать все преимущества движка этой базы данных для безопасного и эффективного хранения информации.

По умолчанию отдельные базы данных приложений изолированы друг от друга, то есть их содержимое может быть использовано только приложением, которое создало ту или иную базу. Однако Источники данных (Content Providers) обеспечивают возможность совместного использования баз данных приложений.

Базы данных и Источники данных рассматриваются в главе 7.

Общие данные и межпрограммное взаимодействие

Android поддерживает три технологии передачи информации из приложения любому другому источнику: уведомления, классы переходов и Источники данных.

Уведомления — это стандартные средства, с помощью которых мобильные устройства что-либо сообщают пользователю. С помощью API вы можете вызывать звуковые сообщения, создавать вибрацию или отображать флеш-сообщения на экране устройства, а также менять статус значков уведомлений в строке состояния. Более подробная информация об этом в главе 9.

Классы переходов — это механизм передачи сообщений внутри приложений и между ними. С их помощью вы можете транслировать нужное действие (например, набор номера на телефоне или редактирование контакта) по всей системе в другие приложения, которые должны его обработать. Классы переходов — весьма важный компонент ядра платформы Android. Более подробно о них вы узнаете в главе 5.

Наконец, вы можете использовать Источники данных, чтобы открыть доступ к базам данных программы. Встроенные приложения, например менеджер контактов, обеспечивают доступ к информации также через Источники данных, так что вы можете создавать программы, которые будут считывать или изменять эти данные. Источники данных рассматриваются в главе 7, где вы узнаете о встроенных провайдерах и научитесь создавать собственные.

Работа с виджетами, Живыми каталогами и Живыми обоями для расширения возможностей стандартного Рабочего стола

Виджеты, Живые каталоги и Живые обои призваны создавать динамические компоненты приложений, которые, с одной стороны, открывают интерактивное окно в программу, а с другой — позволяют отображать важную или меняющуюся в режиме реального времени информацию прямо на Рабочем столе устройства.

Вы как бы создаете динамический ярлык на программу, который дает возможность взаимодействовать с ней непосредственно с Рабочего стола, так что пользователь в режиме реального времени имеет доступ к интересующей информации без запуска программы.

Вы научитесь создавать компоненты программы для Рабочего стола в главе 10.

Расширенная поддержка мультимедиа и 2D/3D-графики

Большие экраны и яркие дисплеи высокого разрешения позволили назвать мобильные телефоны мультимедийными устройствами. Для использования всех возможностей доступного аппаратного обеспечения Android снабдили графическими библиотеками для двухмерного рисования на Холсте и работы с трехмерной графикой посредством OpenGL.

Android включает также комплексные библиотеки для работы со статическими изображениями, видео- и аудиофайлами, в том числе поддержку форматов MPEG4, H.264, MP3, AAC, AMR, JPG, PNG и GIF.

Использование двух- и трехмерной графики рассматривается в главе 15, библиотеки для работы с мультимедиа описаны в главе 11.

Оптимизированное управление памятью и процессами

Управление памятью и процессами в Android немного необычно. Как и платформы Java и .NET, Android использует собственную среду исполнения и виртуальную машину для управления памятью приложения. Но в отличие от двух вышеназванных платформ в Android среда исполнения управляет

временем жизни процессов. Это позволяет повысить отзывчивость приложений путем остановки или принудительного завершения процессов, если необходимо освободить ресурсы для более приоритетных программ.

В этом контексте высокий приоритет имеет программа, с которой в данный момент работает пользователь. В среде, где программы не могут управлять своим временем жизни, важно обеспечить готовность приложений к быстрому завершению с сохранением их отзывчивости, обновления и перезапуска в фоновом режиме, если такая необходимость возникнет.

Более подробную информацию о жизненном цикле приложений, работающих на платформе Android, вы получите в главе 3.

Несколько слов об Альянсе открытых мобильных устройств (Open Handset Alliance, OHA)

Альянс открытых мобильных устройств — это сообщество из более чем 50 компаний, включающее производителей аппаратного и программного обеспечения, а также мобильных операторов. Среди наиболее значительных членов Альянса можно назвать компании Motorola, HTC, T-Mobile и Qualcomm. Вот как формулируют основные идеи OHA участники этого сообщества:

Приверженность открытости, общее видение будущего и конкретные задачи для воплощения мечты в реальность. Ускорение внедрения инноваций в сфере мобильных технологий и предоставление потребителям функциональных, менее дорогих и более продвинутых мобильных устройств. (http://www.openhandsetalliance.com/).

Цель OHA — донести до покупателей свой богатый опыт в сфере разработки программного обеспечения. В связи с этим представлена платформа, идеально подходящая для внедрения инновационных технологий, которая отличается более высокой производительностью и улучшенным качеством по сравнению с существующими. При всем этом, используя данную платформу, ни разработчики ПО, ни производители мобильных устройств не должны платить какие-либо лицензионные отчисления.

На каких устройствах работает Android

Первым мобильным телефоном на платформе Android стал T-Mobile G1, который выпустили в США в октябре 2008 г. К концу 2009 г. было анонсировано или выпущено более 20 мобильных устройств на платформе Android в более чем 26 странах в сетях 32 различных мобильных операторов.

Android — мобильная операционная система, созданная не для одной определенной аппаратной реализации, а для множества разнообразных аппаратных платформ, начиная с телефонов с WVGA-дисплеем и кнопочной клавиатурой и заканчивая устройствами с резистивными сенсорными экранами.

Причиной такой популярности стала более низкая стоимость устройств на платформе Android, которая достигается за счет отсутствия лицензионных платежей или оплат за встроенное ПО. Многие ожидают, что преимущества Android как платформы для разработки функциональных приложений подтолкнет производителей мобильных телефонов к усовершенствованию аппаратной части.

Зачем нужно заниматься разработкой ПО для мобильных устройств

С точки зрения экономики увеличение числа современных мобильных смартфонов и суперфонов (то есть мультифункциональных устройств, включающих не только функции телефона, но и полноценный веб-браузер, камеру, плеер, Wi-Fi и службы геопозиционирования), привело к фундаментальному изменению способа взаимодействия человека с мобильным телефоном и сетью Интернет. Во многих странах количество мобильных телефонов во много раз превосходит число компьютеров. В 2009 году через мобильные телефоны впервые в Интернет вышло большее количество людей, чем через компьютеры.

Рост популярности современных смартфонов вместе с развитием сетей Wi-Fi и появлением доступных тарифных планов с фиксированной платой стали предпосылкой для стремительного развития рынка функциональных мобильных приложений.

То, что мобильные телефоны вошли во все сферы жизни и теперь мы практически не можем без них обходиться, привело к появлению фундаментально новой платформы для разработки приложений, отличной от классических платформ персональных компьютеров. Телефон с микрофоном, камерой, сенсорным экраном, навигатором и датчиками положения стал нашим новым средством восприятия окружающего мира.

В среднем пользователь платформы Android устанавливает и использует около 40 различных приложений, а это свидетельствует об изменении способа взаимодействия людей со своими телефонами. Вам, как разработчику, выпала уникальная возможность создавать динамические, интересные и совершенно новые программы, которые могут стать неотъемлемой частью жизни многих людей.

Для чего нужно заниматься разработкой приложений для Android

Если у вас есть опыт разработки мобильных приложений, не мне говорить вам, что:

- многое из того, что вы можете сделать, уже возможно на платформе Android;
- сделать это будет чрезвычайно непросто.

Android потребует полного изменения парадигмы, так как вы имеете дело с мобильным фреймворком, который ориентирован на современные мобильные устройства и создан программистами для разработчиков.

Благодаря простой и функциональной среде, отсутствию лицензионных отчислений, наличию подробной документации и постоянно растущему сообществу разработчиков Android предоставляет отличную возможность для создания программ, которые меняют цели и способы взаимодействия людей со своими мобильными телефонами.

С коммерческой точки зрения Android:

- не требует какой-либо сертификации для написания программ;
- предоставляет доступ к сервису Android Market, где можно размещать и продавать свои программы;
- исключает какие-либо механизмы контроля предоставляемых вами программ;
- дает возможность разрабатывать версии платформы под собственной торговой маркой, то есть предоставляет вам полный контроль над интерфейсом пользователя.

Какое дальнейшее развитие получит Android

Android ориентирован в первую очередь на разработчиков; Google и OHA уверены, что для предоставления клиентам более совершенного мобильного ПО нужно дать возможность самим пользователям-программистам написать удобные для них программы.

Android — мощная и интуитивно понятная платформа для разработки, и благодаря этому программисты, которые никогда не имели дела с программами для мобильных устройств, могут легко и быстро начать создавать полноценные приложения для Android. Совершенно очевидно, что появление множества инновационных приложений для Android создаст спрос на устройства, на которых они могут запускаться, в особенности если разработчики будут писать приложения для Android только потому, что они *не смогут* писать их для других платформ.

Открытый доступ к ядру операционной системы — это один из факторов, который всегда способствовал развитию ПО и расширению сферы внедрения платформы.

Присущие Интернету открытость и нейтральность за какие-то 10 лет превратили сеть в идеальную платформу для бизнеса с многомиллиардными долларовыми оборотами.

Однако еще до этого свободный характер некоторых операционных систем, таких как Linux или мощный API ОС Windows, определил их стремительное развитие на платформе персональных компьютеров, благодаря чему программирование из единичных увлечений стало распространено повсеместно.

Открытость и функциональность гарантируют, что любой человек с соответствующими интересами может внести свой вклад в общее дело с минимальными затратами.

Что здесь есть такого, чего нет у других

Многие упомянутые выше характеристики, такие как поддержка 3D-графики и встроенные базы данных, присутствуют и в других средах разработки мобильных приложений. Ниже перечислены уникальные, присущие только Android возможности.

- **Приложения Google Map.** Сервис Google Maps for Mobile (карты Google для мобильных телефонов) пользуется огромной популярностью, и Android предлагает возможность управления Google Map из приложений. Map View позволяет отображать, изменять и комментировать карты Google внутри Активностей (Activities), благодаря чему можно строить картографические приложения с интерфейсом, аналогичным Google Maps.
- **Фоновые службы и приложения.** Фоновые службы позволяют программе взаимодействовать с событийной моделью приложений, работая незаметно для вас, в то время когда вы используете другие программы или телефон находится в режиме ожидания, пока он не зазвонит, не начнет вибрировать или подавать световые сигналы, чтобы привлечь внимание. Примеров фоновых приложений можно назвать множество. Это и потоковый музыкальный плеер, и приложение, которое отслеживает биржевые сводки, сообщая о значительных изменениях в вашем портфеле ценных бумаг, и сервис, который меняет мелодию, громкость звонка в зависимости от вашего текущего местоположения, времени суток или того, кто звонит.
- **Общие данные и межпроцессная коммуникация.** Благодаря встроенным в Android классам переходов и Источникам данных приложения могут обмениваться сообщениями, обрабатывать данные или предоставлять общий доступ к ним. Вы также можете использовать

эти механизмы для повышения эффективности работы с данными или функционалом, предоставляемыми встроенными в Android приложениями. Для снижения возможных уязвимостей открытой архитектуры каждый процесс, хранилище данных и файл закрыты для доступа, пока их намеренно не сделают общедоступными другим приложениям. Для этого используется специальная служба безопасности, обеспечивающая полное разделение прав доступа к ресурсам (подробнее о ней в главе 15).

- **Все создаваемые приложения имеют равный статус.** Android не различает встроенные приложения и программы сторонних разработчиков. Благодаря этому пользователи получают широкие возможности изменения интерфейса и функционала своих устройств. Они могут заменить любое встроенное приложение альтернативными, предложенными сторонними разработчиками, которые получают аналогичный доступ к данным и аппаратному обеспечению устройства.
- **Рабочий стол и работа с виджетами, Живыми каталогами, Живыми обоями и панелью быстрого поиска.** Благодаря функционалу виджетов, Живых каталогов и Живых обоев вы можете создавать окна в свои приложения, которые будут располагаться на Рабочем столе телефона. Панель быстрого поиска позволяет включать результаты поиска ваших приложений в поисковую систему телефона.

Изменение подхода к разработке мобильных приложений

Существующие мобильные платформы разработки создали ауру эксклюзивности вокруг программирования мобильных приложений. Многие мобильные телефоны остаются по сути такими же, какими они были в первый день покупки, благодаря то ли дизайну, то ли воздействию цены и необходимости лицензирования разрабатываемых программ.

В противоположность этому Android позволяет или, можно сказать, даже мотивирует делать радикальные изменения. Устройства под управлением Android поставляются с набором стандартных приложений, которые необходимы пользователям на их новом телефоне, но главная мощь платформы заключается в возможности радикального изменения интерфейса и функционала.

Android предоставляет разработчикам поистине уникальные возможности. Все без исключения приложения на этой платформе — неотъемлемая часть телефона, а не просто программы, которые работают изолированно поверх приложений ядра. Теперь вы создаете не просто приложения для устройств с небольшими экранами и малой мощностью, а разрабатываете мобильные программы, которые изменяют способ взаимодействия людей с их телефонами.

В то время пока Android будет конкурировать с существующими или будущими мобильными платформами разработки как открытый фреймворк, пользователи станут отдавать предпочтение наиболее удобным средам разработки. В любом случае бесплатность и открытый подход к созданию мобильных приложений в сочетании с полным доступом к ресурсам телефона — гигантский шаг в правильном направлении.

Знакомство с фреймворком разработчика

«Рекламная кампания» закончена, теперь можно приступать к рассмотрению процесса разработки приложений для Android. Язык программирования приложений для платформы Android — Java. Однако они исполняются не на классической Java VM, а на специальной виртуальной машине Dalvik. Чуть ниже в данной главе будет представлен фреймворк.

Сначала мы расскажем о технических особенностях программного стека Android затем опишем среду разработки и познакомим с библиотеками Android, а в конце и с виртуальной машиной Dalvik.

Каждое приложение для Android функционирует в отдельном процессе внутри собственного экземпляра машины Dalvik. Вся ответственность за память и управление процессами возлагается на Android, который останавливает или убивает процессы, если нужно освободить ресурсы.

Dalvik и Android находятся на вершине ядра Linux, которое занимается низкоуровневым взаимодействием с аппаратным обеспечением, включая работу драйверов и управление памятью. При этом набор встроенного API позволяет получить доступ ко всем службам, функционалу и аппаратной начинке.

Что в комплекте

Среда разработки приложений для Android включает все необходимое для создания, тестирования и отладки программ. Итак, что входит в состав скачиваемого комплекта?

- **API-платформы Android.** Ядро среды разработки — библиотеки API, которые обеспечивают программисту доступ к стеку платформы Android. Эти же самые библиотеки используются компанией Google для написания встроенных в Android приложений.
- **Инструменты разработки.** Вы можете преобразовать исходный код в исполняемые приложения для Android. В состав среды разработки входят инструменты, которые позволяют компилировать и отлаживать приложения. Более подробную информацию об инструментах разработки читайте в главе 2.

- **Менеджер виртуальных устройств и эмулятор.** Эмулятор Android — это полностью интерактивный эмулятор устройств, включающий несколько альтернативных вариантов интерфейса. Эмулятор запускается внутри виртуального устройства Android, которое моделирует аппаратную конфигурацию определенной модели телефона. С помощью эмулятора вы можете увидеть, как приложения будут выглядеть и функционировать на реальном устройстве под управлением Android. Все программы здесь запускаются внутри виртуальной машины Dalvik, программный эмулятор создает для них идеальное окружение, которое не зависит от особенностей той или иной аппаратной начинки и представляет собой лучшую независимую среду для тестирования, чем любая конкретная аппаратная реализация.
- **Полный набор документации.** Среда разработки включает расширенную справочную информацию с примерами кода, в которой описывается, что входит в каждый программный пакет и класс и как их можно использовать. В дополнение к этому в справочной документации содержится стартовый курс для начинающих, а также подробное описание основ разработки программ для Android.
- **Примеры кода.** В среде разработки вы найдете примеры программ, которые демонстрируют некоторые возможности Android, а также несколько простых приложений, посвященных определенным функциям API.
- **Онлайн-поддержка.** Группы Google по адресу http://developer.android.com/resources/community-groups.html – это форумы разработчиков, на которых часто появляются сообщения от команд инженеров и специалистов компании Google. Ресурс StackOverflow по адресу http://www.stackoverflow.com/questions/tagged/android также стал популярным — здесь публикуются вопросы, посвященные Android.

Для тех, кто использует популярную среду разработки Eclipse, Android выпустил специальный плагин, который упрощает процесс создания проекта и тесно связывает Eclipse с эмулятором и отладочными инструментами Android. Информация о плагине ADT приводится в главе 2.

Программный стек Android

Программный стек Android состоит из элементов, показанных на рис. 1.1. Их подробное описание приводится ниже. Упрощенно их можно представить как комбинацию ядра Linux и набора библиотек C/C++, которые доступны в фреймворке приложения. Последний обеспечивает управление и функционирование рабочей среды и приложений.

- **Ядро Linux**. Работу системных служб (драйверы устройств, управление процессами и памятью, питанием, безопасность, сетевые службы) обеспечивает ядро Linux версии 2.6. Оно также отвечает за уровень абстракции между аппаратной начинкой и остальной частью программного стека.

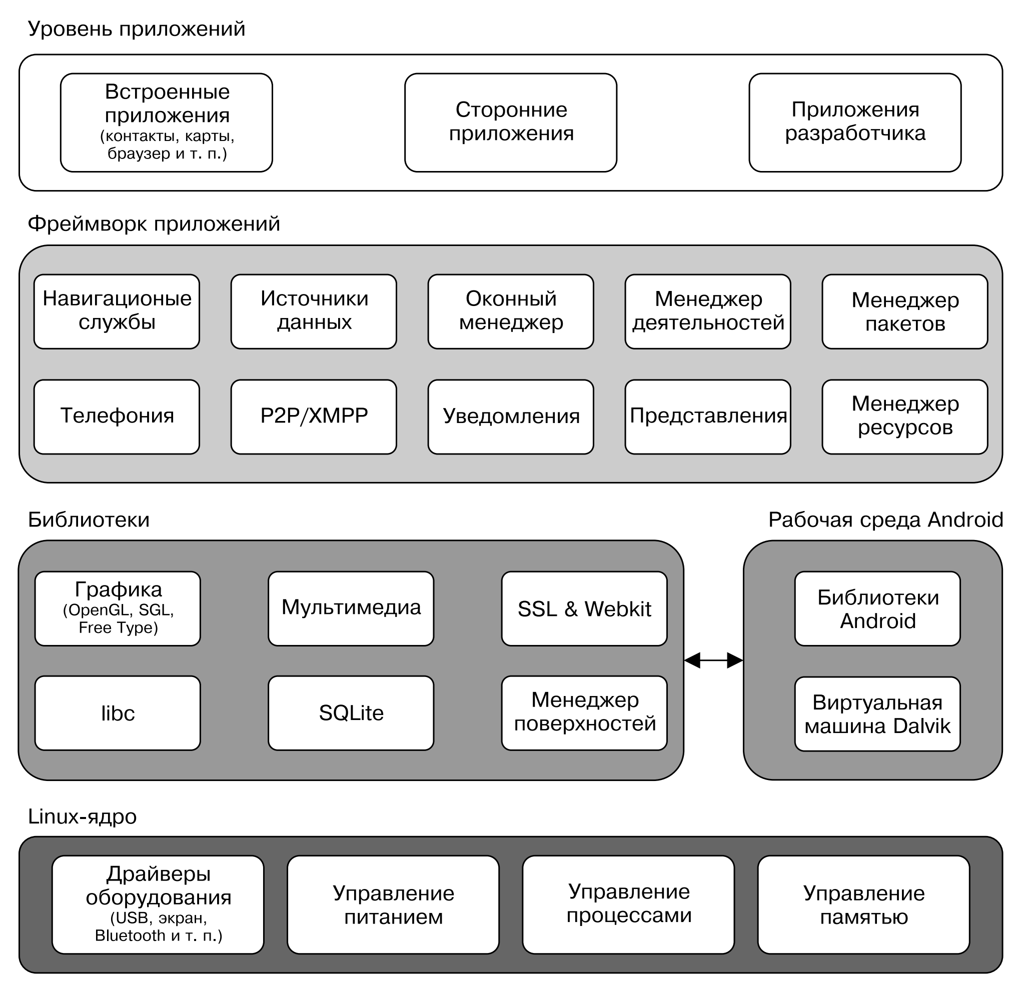


Рис. 1.1.

- **Библиотеки.** Android включает разнообразные системные библиотеки C/C++ (например, SSL и libc), которые работают поверх ядра. Среди них можно выделить:
  - библиотеку для работы с мультимедиа, которая обеспечивает проигрывание аудио- и видеофайлов;
  - менеджер интерфейса, отвечающий за управление отображением;
  - графические библиотеки, такие как SQL и OpenGL, для работы с 2D- и 3D-графикой;
  - библиотеку SQLite, обеспечивающую работу встроенных баз данных;
  - SSL и WebKit для работы встроенного веб-браузера и обеспечения интернет-безопасности.

- **Рабочая среда Android.** Особенным телефон на платформе Android делает не столько мобильная версия ОС Linux, сколько рабочая среда Android. Она включает в себя библиотеки ядра и виртуальную машину Dalvik и обеспечивает функционирование программ, а вместе с библиотеками формирует основу фреймворка приложений.
  - Библиотеки ядра. Хотя приложения для Android разрабатываются на языке Java, Dalvik — это не виртуальная Java-машина. Библиотеки ядра Android обеспечивают основную функциональность библиотек ядра Java, а также присущий Android уникальный функционал.
  - Виртуальная машина Dalvik. Dalvik — это виртуальная машина на основе регистров, которая оптимизирована таким образом, чтобы на устройстве можно было запускать несколько приложений одновременно. В ее основе ядро Linux, которое обеспечивает работу потоков и низкоуровневое управление памятью.
  - Фреймворк приложений. Фреймворк включает набор классов, которые используются для разработки приложений. Он также предоставляет обобщенные абстрактные классы для доступа к оборудованию и обеспечивает управление пользовательским интерфейсом и ресурсами приложения.
  - Уровень приложений. Все программы, как встроенные, так и сторонние, разрабатываются на уровне приложений с использованием одних и тех же библиотек API. Уровень приложений функционирует внутри рабочей среды Android, используя классы и службы, открытые для доступа на этом уровне.

Виртуальная машина Dalvik

Один из ключевых компонентов Android — виртуальная машина (ВМ) Dalvik. Вместо классической виртуальной Java-машины, такой как Java ME (Java Mobile Edition), Android использует собственную ВМ, разработанную для обеспечения эффективной работы нескольких приложений на одном устройстве.

В основе ВМ Dalvik ядро Linux, которое обеспечивает работу таких низкоуровневых функций, как безопасность, потоки, управление процессами и памятью. Вы можете также писать приложения C/C++, которые будут работать непосредственно на базовом уровне ОС Linux. Хотя такая возможность и существует, необходимости в этом нет никакой.

Если для приложения важны присущие C/C++ скорость и эффективность работы, Android предоставляет доступ к нативной среде разработки (NDK). Она позволяет разрабатывать библиотеки C++ с использованием библиотек libc и libm, а также обеспечивает нативный доступ к OpenGL.

ПРИМЕЧАНИЕ

Эта книга посвящена исключительно разработке с помощью SDK (среда разработки программ) приложений, которые запускаются на ВМ Dalvik. Если вы интересуетесь ядром Linux или системными программами на C/C++ для Android, если хотите вносить изменения в код виртуальной машины или каких-либо других системных программ, предлагаем обратиться на сайт Android Internals Google Group по адресу http://groups.google.com/group/android-internals.

Хотя специалисты рекомендуют в случае необходимости использовать NDK, информация о работе с этой средой разработки в данную книгу не была включена.

Доступ к устройствам и системным службам Android осуществляется через виртуальную машину Dalvik, которая считается промежуточным слоем. Благодаря использованию ВМ для выполнения кода программы управляющей системы разработчики получают в свое распоряжение уровень абстракции, который позволяет им не беспокоиться об особенностях конструкции того или иного устройства.

ВМ Dalvik запускает исполняемые файлы, формат которых оптимизирован под минимальное использование памяти. Вы создаете исполняемый файл с расширением `.dex` путем трансформирования компилированных классов, написанных на языке Java, используя для этого инструменты, входящие в состав среды разработки. В следующей главе вы узнаете о создании исполняемых файлов формата Dalvik.

Архитектура Android-приложений

В основе архитектуры Android идея многократного использования компонентов, благодаря чему вы можете публиковать и делиться Активностями, Сервисами (Services) и данными с другими приложениями с возможностью управления доступом с помощью политик безопасности, которые вы назначаете прямо на месте.

Аналогичный механизм, с помощью которого вы можете создать собственный менеджер контактов или программу для набора номера, позволяет предоставлять другим программистам свои компоненты — на их основе они смогут создавать новые пользовательские интерфейсы или функциональные расширения, а также строить свои приложения.

Перечислим службы приложений, которые можно назвать базовыми составляющими архитектуры всех приложений для Android, а также основой фреймворка, который вы будете использовать в собственных программах.

- **Менеджер Активностей.** Контроль за жизненным циклом Активностей (Activity), включая управление стеком Активностей, описывается в главе 3.

- **Представления.** Используются при создании пользовательских интерфейсов для ваших Активностей (глава 4).
- **Менеджер уведомлений.** Обеспечивает работу унифицированных ненавязчивых уведомлений для пользователей (глава 9).
- **Источники данных.** Позволяют приложениям открывать доступ к данным (глава 7).
- **Менеджер ресурсов.** Обеспечивает отображение некодированных ресурсов, таких как текстовые строки или изображения (глава 3).

Библиотеки Android

Android предоставляет набор API-библиотек для разработки приложений. Мы не будем приводить их полный список. Всем, кому интересно, рекомендуем обратиться по адресу: http://developer.android.com/reference/packages.html, где вы найдете полный список пакетов, включенных в состав среды разработки Android.

Android задуман для работы на множестве разнообразных мобильных устройств, поэтому примите во внимание, что совместимость и функционирование некоторых продвинутых или дополнительных API-функций может значительно варьироваться на том или ином устройстве.

Резюме

В этой главе мы рассказали, что при всей стремительности развития аппаратного обеспечения современных мобильных устройств наметилось сильное отставание программного обеспечения. Сложные в использовании среды разработки, специфический для того или иного оборудования API, а также отсутствие открытости сильно сдерживали инновационное развитие мобильного ПО.

Android открывает перед разработчиками широкие возможности по созданию инновационных программ для мобильных устройств без каких-либо ограничений (что характерно для существующих закрытых мобильных фреймворков).

Мы рассказали о программном стеке Android, который включает не только уровень приложений и инструменты для разработки, но и ВМ Dalvik, собственную рабочую среду, библиотеки ядра и ядро Linux. Все это предоставляется с открытыми исходными кодами.

Вы также узнали:

- как устройства с постоянно совершенствующимися аппаратными возможностями создали спрос на инструменты, которые обеспечивают полный доступ к этим возможностям;

- о некоторых функциональных особенностях платформы Android, доступных разработчикам, например, поддержке встроенных карт, доступе к оборудованию, фоновых службах, обмене сообщениями между процессами, общих базах данных и поддержке 2D- и 3D-графики;
- что все приложения на Android имеют одинаковый статус, а это позволяет пользователям заменить одно приложение на другое, включая встроенные системные программы;
- что среда разработки для Android включает инструменты разработчика, библиотеки API и обширную документацию.

В следующей главе мы приступим к разработке, загрузим и установим инструменты-разработчики приложений для Android и настроим среду программирования под управлением Eclipse.

Прежде чем вы начнете свое первое приложение для Android, мы научим вас работать с плагинами среды разработки Android для поэтапного создания приложений, тестирования и отладки.

После изучения этапов разработки приложений расскажем о различных типах программ, которые можно создавать. Вы также узнаете о некоторых аспектах дизайна, которые необходимо учитывать при разработке приложений для мобильных устройств.

Глава 2
ПРИСТУПАЕМ К РАБОТЕ

Содержание главы

- Установка инструментов разработчика Android, построение среды разработки, отладка проектов.
- Требования к дизайну и вопросы оптимизации для увеличения скорости и эффективности работы, создание интерфейсов для маленьких экранов и передача мобильных данных.
- Работа с виртуальными устройствами Android, эмулятор и инструменты разработки.

Чтобы начать разработку собственных приложений для Android, вам понадобится установочный пакет Android SDK и средства разработки Java-приложений. Если вам хочется помучиться, можете поставить Java IDE — Eclipse отличается хорошей поддержкой, что немного облегчает разработку.

Существуют версии SDK, Java и Eclipse для различных платформ — Windows, MacOS и Linux, таким образом, вы можете создавать приложения для Android на любой удобной для вас ОС. Инструменты разработчика и эмулятор функционируют на всех трех платформах, и поскольку приложения Android запускаются на виртуальной машине, никакая из операционных систем не будет давать существенного выигрыша для программиста.

Код приложений для Android пишется с соблюдением синтаксиса Java, библиотеки ядра Android поддерживают большинство функций ядра API Java. Перед исполнением проекты должны быть переведены в байт-код ВМ Dalvik. В результате вы можете использовать все преимущества Java, при этом приложения будут оптимизированы для работы на виртуальной машине, которая приспособлена к требованиям мобильных устройств.

Пакет установки SDK включает все библиотеки Android, набор документации и примеры приложений. В него также входят инструменты, которые помогают разрабатывать и отлаживать приложения. Среди них можно на-

звать эмулятор Android для запуска приложений и службу мониторинга отладки Dalvik (DDMS).

Изучая данную главу, вы скачаете пакет установки, настроите среду разработчика, создадите два новых приложения, запустите их на исполнение и произведете отладку с помощью DDMS на эмуляторе, работающем на базе виртуального устройства Android.

Если вы уже разрабатывали мобильные приложения, то знаете, что небольшой форм-фактор телефонов, ограничения по питанию и памяти предъявляют особые требования к дизайну приложений. И даже если вы новичок, должно быть очевидно, что отдельные вещи, которые считаются само собой разумеющимися на компьютере или в Интернете, не будут аналогично работать на мобильном телефоне.

Помимо аппаратных ограничений необходимо учитывать пользовательское окружение. Мобильные телефоны используются на ходу, часто они выступают отвлекающими факторами и не всегда в центре внимания, таким образом, ваши приложения должны быть быстрыми, отзывчивыми и простыми в изучении.

В данной главе мы дадим ценные рекомендации, которые позволят учесть при разработке пользовательский фактор и преодолеть проблемы, связанные с ограниченными аппаратными ресурсами. Мы не будем охватывать большой объем информации, а остановимся на работе с инструментами разработчика Android с соблюдением рекомендаций по дизайну мобильных приложений.

Разработка приложений для Android

Инструменты разработчика Android включают все необходимые программы и API-библиотеки, которые понадобятся для разработки совершенных и функциональных мобильных приложений. Одной из основных трудностей в Android, как и в любой другой новой для вас среде разработки, станет необходимость изучения функционала и ограничений API.

Если у вас есть опыт разработки Java-программ, то заметите, что методы, синтаксис и грамматика Java-кода могут быть использованы и на платформе Android. Тем не менее, некоторые специфические способы оптимизации могут показаться немного противоречивыми.

Если у вас нет опыта написания Java-кода, но вы знакомы с каким-либо объектно-ориентированным языком программирования (например, C#), новый язык покажется простым. Основная мощь Android в его API, а не в языке Java, поэтому даже если вы не сталкивались со специфическими Java-классами, то не почувствуете себя ущемленным.

С чего начать

Поскольку приложения для Android исполняются на виртуальной машине Dalvik, вы можете писать программы на любой платформе, где можно установить среду разработки. К таковым относятся:

- Microsoft Windows (XP и выше);
- Mac OS X 10.4.8 или выше (только для чипсетов Intel);
- Linux.

Перед началом работы необходимо скачать и установить следующие пакеты:

- SDK Android (инструменты разработчика для Android);
- Java Development Kit (инструменты разработки Java-приложений) (JDK) версии 5 или 6.

Последнюю версию JDK можно скачать с сайта компании Sun по адресу http://java.sun.com/javase/downloads/index.jsp.

ПРИМЕЧАНИЕ

Если вы уже установили JDK, убедитесь, что версия данного пакета соответствует приведенным выше требованиям. Обращаем особое внимание, что для работы недостаточно иметь Java runtime environment (JRE, Исполняющая среда Java).

Скачиваем и устанавливаем SDK

SDK для Android имеет открытый характер. Вы не должны ничего платить за скачивание или использование API. Компания Google не требует платы (или отчета) за размещение ваших программ на сервисе Android Market или где-либо еще.

Самую последнюю версию SDK можно скачать на сайте для разработчиков Android по адресу http://developer.android.com/sdk/index.html.

ПРИМЕЧАНИЕ

При написании этой книги мы использовали SDK версии 2.1 r1.

SDK представляет собой ZIP-файл, который содержит самую последнюю версию инструментов разработки Android. Распакуйте содержимое файла в новую папку (запомните путь к данной папке, он вам понадобится позже).

Прежде чем начать работу, необходимо установить по меньшей мере одну платформу SDK. Для этого в среде Windows запустите файл `SDK Setup.exe`; если у вас MacOS или Linux — исполняемый файл `android`, который расположен в подпапке tools. В появившемся окне выберите слева пункт Available Packages, после чего справа в области Sources, Packages and Archives

отметьте пакеты платформы SDK, которые хотите установить. Выбранные файлы скачаются в папку установки SDK и будут включать библиотеки API, документацию и несколько примеров приложений.

Все примеры и пошаговые инструкции в данной книге рассчитаны на работу со средой разработки Eclipse с интегрированным в нее плагином ADT (Инструмент разработки для Android). Тем не менее, это необязательное требование — вы можете использовать любой удобный для вас текстовый редактор или Java IDE, а также программы в составе SDK для компиляции, тестирования и отладки фрагментов кода или примеров приложений.

Если вы планируете воспользоваться первым вариантом, обратитесь к следующему разделу, где мы расскажем о настройке Eclipse и плагина ADT. Для тех, кто собирается программировать без Eclipse и ADT, ниже дается описание инструментов разработки, которые входят в состав SDK.

ПРИМЕЧАНИЕ

Примеры приложений в составе SDK неплохо описаны в документации, а их исходный код — прекрасный материал, иллюстрирующий образцовые приложения для Android. После настройки среды разработки рекомендуем изучить исходный код этих примеров.

Разработка в среде Eclipse

Работа в Eclipse вместе с плагином ADT дает существенные преимущества.

Eclipse — это среда разработки с открытыми исходными кодами, которая, как правило, используется для программирования Java-приложений. Вы можете скачать ее с сайта по адресу www.eclipse.org/downloads/ и установить на любой платформе, поддерживаемой Android (Windows, MacOS и Linux).

Есть множество версий Eclipse, однако для Android рекомендуется конфигурация Eclipse 3.4 или 3.5 (Galileo):

- плагин Eclipse JDT;
- WST.

WST и плагин JDT входят в состав большинства версий Eclipse.

Чтобы установить Eclipse, распакуйте скачанный архив в новую папку и запустите исполняемый файл `eclipse`. При первом запуске создайте новую рабочую среду для ваших проектов.

Работа с плагином для Eclipse

ADT-плагин упрощает разработку приложений для Android путем интеграции инструментов разработчика, включая эмулятор и конвертор файлов из формата `.class` в формат `.dex` непосредственно в среду Eclipse. Хотя

плагин ADT необязателен, он существенно облегчает и ускоряет разработку, тестирование и отладку приложений.

Плагин добавляет в среду Eclipse следующие модули:

- мастер проектов Android, который помогает создавать новые проекты и включает стандартный шаблон приложения;
- основанный на формах манифест, шаблон и редактор ресурсов, которые помогают создавать, редактировать и проверять ваши XML-ресурсы;
- автоматическое построение проектов для Android, конвертацию исходных кодов в исполняемые файлы (`.dex`), упаковку в файлы пакетов (`.apk`) и установку пакетов в виртуальные машины Dalvik;
- менеджер виртуальных устройств Android, позволяющий создавать и управлять эмуляторами с виртуальными устройствами, на которых имитируется запуск специальных релизов ОС Android с присутствующими на устройстве ограничениями памяти;
- эмулятор Android с возможностью настройки его интерфейса и изменения установок сети, а также способностью имитировать входящие звонки и SMS-сообщения;
- службу мониторинга отладки Dalvik (DDMS) с функцией перенаправления портов, стеком, кучей и возможностью просмотра потоков, информации о процессах и функцией снятия скриншотов;
- доступ к устройству или файловой системе эмулятора, благодаря чему можно перемещаться по дереву папок и передавать файлы;
- функцию отладки на этапе выполнения, благодаря чему можно устанавливать точки остановки и просматривать стеки вызовов;
- журнал событий Android и ВМ Dalvik, а также выводимый в консоль список сообщений.

На рис. 2.1 показана работа службы DDMS в среде разработки Eclipse, где установлен плагин ADT.

Установка ADT-плагина

Установите набор инструментов разработчика следующим образом:

1. В меню Eclipse выполните команду Help ▸ Install New Software.
2. В появившемся диалоговом окне в поле Work With введите адрес https://dl-ssl.google.com/android/eclipse/ и нажмите Enter.
3. Eclipse запустит поиск плагина ADT. Все найденные плагины будут отображаться в поле ниже, как показано на рис. 2.2. Отметьте пункт Developer Tools и нажмите кнопку Next.
4. Eclipse начнет скачивать плагин. После этого убедитесь, что отмечены оба плагина (Android DDMS и Android Developer Tools) и нажмите Next.

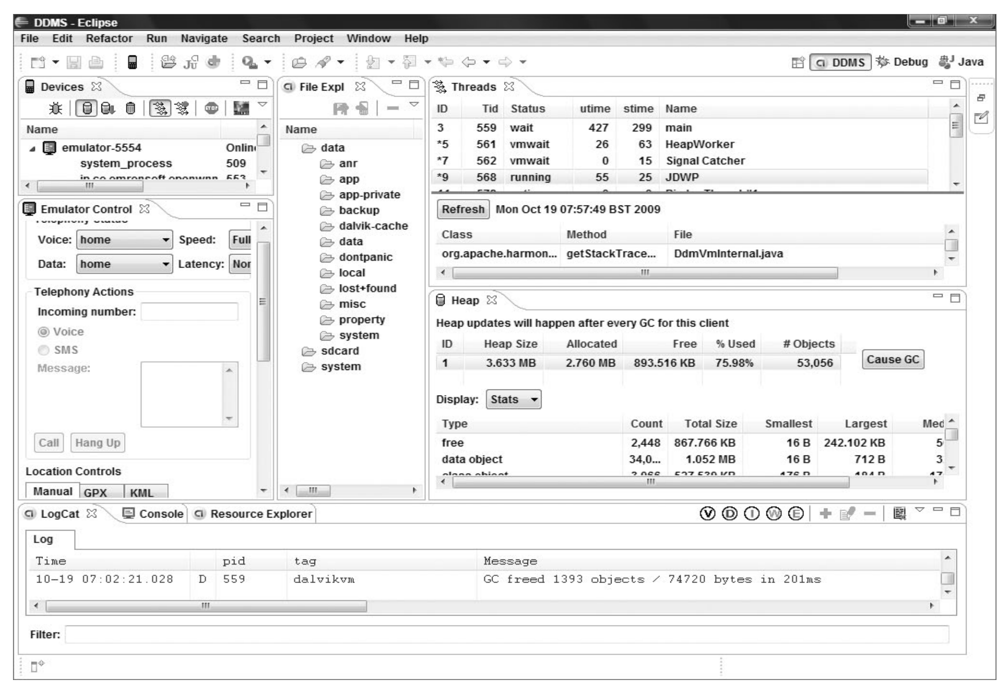

DDMS - Eclipse
File Edit Refactor Run Navigate Search Project Window Help
DDMS Debug Java
Devices
Name
emulator-5554
system_process
509
Emulator Control
Voice: home
Speed: Full
Data: home
Latency: Non
Telephony Actions
Incoming number:
Voice
SMS
Message:
Call
Hang Up
Location Controls
Manual
GPX
KML
File Expl
Name
data
anr
app
app-private
backup
dalvik-cache
data
dontpanic
local
lost+found
misc
property
system
sdcard
system
Threads
ID Tid Status utime stime Name
3 559 wait 427 299 main
*5 561 vmwait 26 63 HeapWorker
*7 562 vmwait 0 15 Signal Catcher
*9 568 running 55 25 JDWP
Refresh
Mon Oct 19 07:57:49 BST 2009
Class Method File
org.apache.harmon... getStackTrace... DdmVmInternal.java
Heap
Heap updates will happen after every GC for this client
ID Heap Size Allocated Free % Used # Objects
1 3.633 MB 2.760 MB 893.516 KB 75.98% 53,056
Cause GC
Display: Stats
Type Count Total Size Smallest Largest
free 2,448 867.766 KB 16 B 242.102 KB
data object 34,0... 1.052 MB 16 B 712 B
LogCat
Console
Resource Explorer
Log
Time pid tag Message
10-19 07:02:21.028 D 559 dalvikvm GC freed 1393 objects / 74720 bytes in 201ms
Filter:

Рис. 2.1.

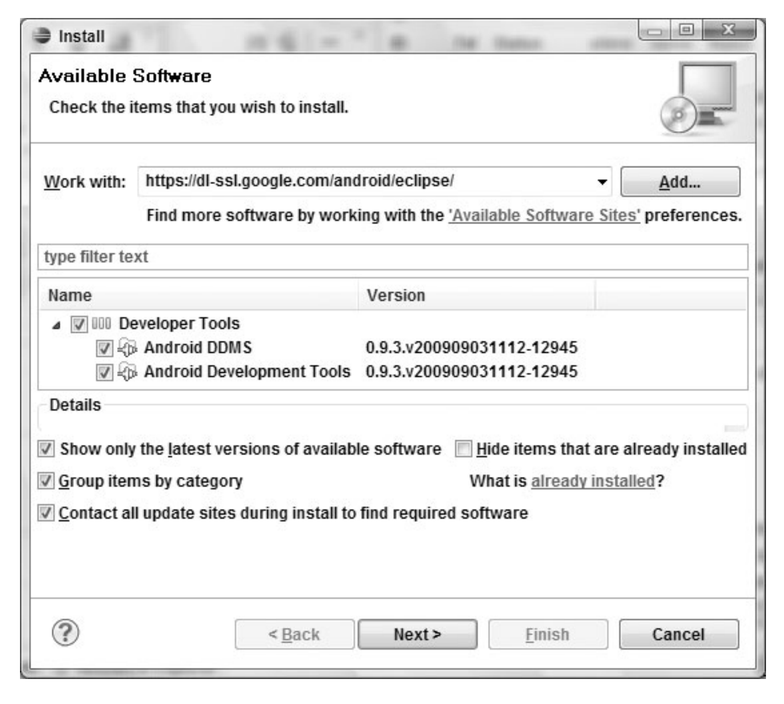

Install
Available Software
Check the items that you wish to install.
Work with: https://dl-ssl.google.com/android/eclipse/
Add...
Find more software by working with the 'Available Software Sites' preferences.
type filter text
Name
Version
Developer Tools
Android DDMS 0.9.3.v200909031112-12945
Android Development Tools 0.9.3.v200909031112-12945
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel

Рис. 2.2.

5. Прочитайте и согласитесь с условиями лицензионного соглашения (I accept the terms of the license agreements), нажмите кнопку Next, а затем Finish. Если плагин ADT окажется неподписанным, вы увидите соответствующее предупреждение перед продолжением установки.

6. После окончания установки нужно перезапустить Eclipse и задать некоторые настройки ADT. Закройте и вновь запустите Eclipse, выберите в меню пункты Window ▸ Preferences или Eclipse ▸ Preferences, если работаете на платформе MacOS.

7. В области слева выберите пункт Android.

8. Нажмите кнопку Browse и укажите путь к папке, в которую вы распаковали архив со средой разработки Android, затем нажмите кнопку Apply. В списке снизу появятся доступные модули среды разработки, как показано на рис. 2.3. Нажмите OK, чтобы завершить установку.

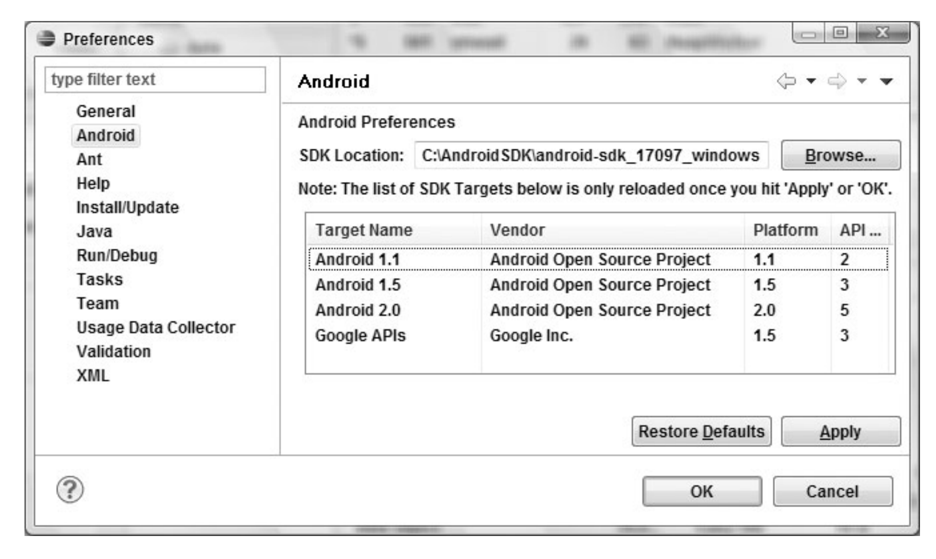


Рис. 2.3.

ПРИМЕЧАНИЕ

Если вы скачали новую версию среды разработки Android, разместите ее в другой директории. После этого вновь понадобится произвести описанную выше настройку, чтобы указать плагину ADT на новую среду разработки, с помощью которой он будет строить приложения.

Обновление плагина

Поскольку среда разработки Android постоянно совершенствуется, появляются все новые обновления для плагина ADT. Во многих случаях процесс установки обновлений очень прост:

1) выполните команду Help ▸ Check for Updates;

2) если будут обнаружены какие-либо обновления для ADT, их список отобразится в результатах поиска. Выберите нужные пакеты обновлений и нажмите Install.

ПРИМЕЧАНИЕ

Иногда вы не сможете использовать встроенную систему обновлений для установки новой версии плагина. В этом случае понадобится удалить предыдущую и установить новую. Для этого воспользуйтесь инструкциями из предыдущего раздела.

Создаем первое приложение для Android

Вы уже скачали инструменты разработчика, установили Eclipse и подключили плагин — теперь готовы к программированию приложений для Android. Мы начнем с создания нового проекта и настройки конфигурации компилятора и отладчика Eclipse.

Создание нового проекта Android

Для создания нового проекта воспользуйтесь Менеджером новых проектов Android.

1. Выберите пункты File ▸ New ▸ Project.
2. В списке типов приложений выберите Android Project и нажмите кнопку Next.
3. В появившемся окне (рис. 2.4) необходимо указать информацию о новом проекте. В поле Project name укажите имя файла проекта. Поле Package name используется для указания названия пакета Java; параметр Create Activity — для указания имени класса, который станет стартовой Активностью; поле Application name можно использовать для назначения понятного имени программы. Min SDK Version позволяет задавать минимальную версию Android, с которой будет совместимо приложение.

ПРИМЕЧАНИЕ

При определении совместимой версии SDK нужно выбрать между расширенной функциональностью новых версий и совместимостью ваших приложений с большим количеством устройств под управлением Android. Приложение будет доступно через сервис Google Android Market на любом устройстве, где установлена выбранная вами или более высокая версия.

Android 1.6 (Donut) — это версия 4. На момент публикации книги в большинстве работающих на Android устройств установлена как минимум четвертая версия. SDK 2.0 (Eclair) — это версия 5, а 2.1 — версия 7.

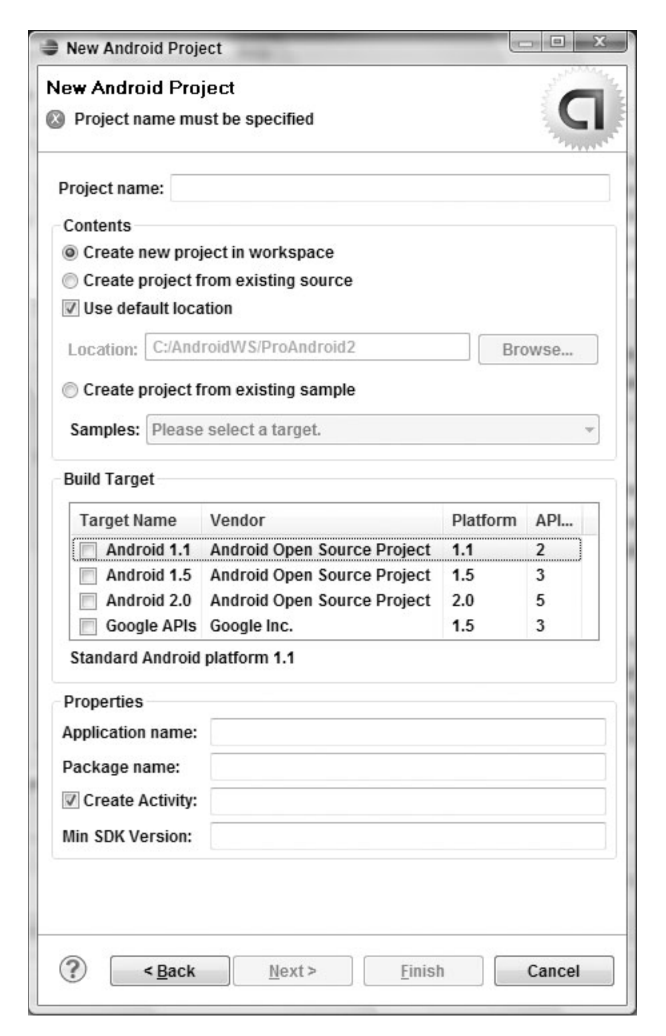


Рис. 2.4.

4. После указания необходимой информации о проекте нажмите Finish.

   Если вы выбрали параметр Create Activity, плагин ADT создаст новый проект, включающий класс, происходящий от класса `Активности`. Вместо пустого проекта в шаблоне по умолчанию будет реализован функционал базового приложения Hello World («Здравствуй, мир»). Перед редактированием кода проекта рекомендуем настроить запуск и отладку.

Сохранение конфигурации запуска приложения

Конфигурация запуска включает настройки компиляции и отладки приложений. Они позволяют управлять следующими функциями:

- выбор проекта и Активности для запуска;
- управление виртуальным устройством и эмулятором;
- настройки ввода и вывода (включая параметры консоли по умолчанию).

Вы можете использовать различные конфигурации для запуска и отладки. Чтобы сохранить конфигурацию запуска для приложения Android, сделайте следующее.

1. В меню Eclipse выполните команду Run ▸ Run Configurations или Debug Configurations.
2. В списке типов проектов выберите пункт Android Application, щелкните на нем правой кнопкой мыши и выберите New.
3. Укажите название для текущей конфигурации. Вы можете создавать несколько конфигураций для каждого проекта. Выбирайте понятные названия, чтобы идентифицировать тот или иной набор параметров.
4. Теперь вы можете приступать к настройке запуска приложения. Первая вкладка (Android) позволяет выбирать проект для компиляции, а также Активность, которая будет запускаться во время компиляции (или отладки) приложения. На рис. 2.5 показаны настройки ранее созданного проекта.

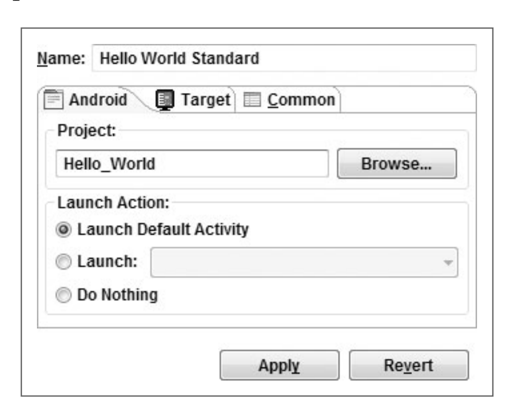


Рис. 2.5.

5. На вкладке Target, которая показана на рис. 2.6, вы можете выбрать виртуальное устройство по умолчанию, на котором будет запускаться приложение. Также можно использовать параметр Manual, при этом при каждой компиляции/отладке нужно выбирать устройство или AVD. На этой вкладке можно изменить сетевые настройки эмулятора, очистить пользовательские данные или отключить анимацию, которая проигрывается во время запуска виртуального устройства. В поле

Command Line Options при необходимости можно указать дополнительные параметры запуска эмулятора.

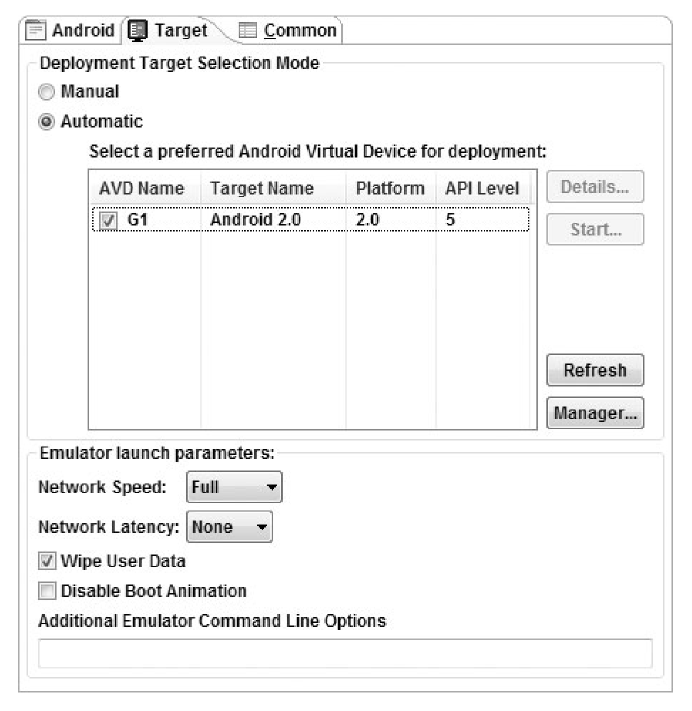


Рис. 2.6

ПРИМЕЧАНИЕ

Среда разработки Android не включает виртуальную машину по умолчанию. Вам понадобится создать виртуальную машину перед компиляцией или отладкой приложений с использованием эмулятора. Если показанный на рис. 2.6 список виртуальных машин пуст (Select a preferred Android Virtual Device for deployment), нажмите кнопку Manage — откроется среда разработки и Менеджер виртуальной машины. Создайте ее. Далее в главе этот процесс описан подробнее.

6. На вкладке Common есть некоторые дополнительные настройки.
7. Нажмите кнопку Apply. Конфигурация для запуска приложения сохранится.

Компиляция и отладка приложений

Вы создали первый проект и сохранили конфигурации компиляции и отладки. Прежде чем делать что-то еще, давайте проверим правильность установки и настройки среды разработки. Для этого попробуем скомпилировать и отладить ваш проект Hello World.

В меню Run выберите пункт Run или Debug — для запуска приложения в соответствии с сохраненными настройками либо Run Configurations, либо Debug Configurations — для выбора конкретной сохраненной конфигурации.

Если вы используете плагин ADT, компиляция или отладка вашего приложения будет происходить следующим образом:

- текущий проект скомпилируется в исполняемый файл платформы Android с расширением `.dex`;
- исполняемый файл и внешние ресурсы упакуются в файл пакета Android с расширением `.apk`;
- запустится выбранное виртуальное устройство (если вы выбрали AVD, и он в данный момент не запущен);
- приложение установится на целевое устройство;
- приложение запустится.

Для отладки используется отладчик Eclipse, в нем можно задавать точки остановки для отладки кода ваших программ.

Если все выполнено правильно, вы увидите новую Активность, запущенную в эмуляторе (рис. 2.7).



Рис. 2.7.

Изучаем код приложения Hello World

Давайте вернемся назад и посмотрим на код вашего первого приложения для Android.

Активность — это базовый класс для визуальных и интерактивных компонентов вашего приложения. Проще говоря, он соответствует классу Form в традиционных языках программирования. В листинге 2.1 приведен код класса на основе Активности. Обратите внимание, что он наследует класс Активности, перекрывая метод onCreate.

Листинг 2.1. Hello World

```
package com.paad.helloworld;

import android.app.Activity;
import android.os.Bundle;
public class HelloWorld extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

В приведенном шаблоне не хватает макета визуального интерфейса. В Android визуальные компоненты называются Представлениями (Views), которые соответствуют компонентам традиционных языков программирования.

В автоматически созданном шаблоне приложения Hello World метод onCreate перекрывается, при этом в его новой реализации вызывается метод setContentView, который выводит пользовательский интерфейс путем подключения ресурса макета, как это показано в листинге:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
}
```

Ресурсы проектов Android хранятся в папке res, которая находится в дереве папок вашего проекта. Она включает подпапки drawable, layout и values. Плагин ADT интерпретирует эти ресурсы для обеспечения доступа к ним на этапе разработки через переменную R (глава 3).

В листинге 2.2 приводится код макета пользовательского интерфейса из файла `main.xml`, который создается шаблоном проекта Android.

Листинг 2.2. Макет проекта Hello World

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
    android:text="Hello World, HelloWorld"
  />
</LinearLayout>
```

Описание и реализация пользовательского интерфейса в XML-файле — наиболее предпочтительный способ реализации интерфейсов в приложениях, поскольку таким образом вы отделяете логику программы от ее визуального представления.

Чтобы получить доступ к элементам интерфейса в коде своих программ, необходимо добавить атрибут идентификатора к объявлениям этих элементов в XML-файле. Вы можете использовать метод findViewById для получения ссылки на каждый из обозначенных таким способом элементов. В приведенном ниже отрывке XML-кода продемонстрировано добавление атрибута ID к виджету Text View в шаблоне приложения Hello World:

```
<TextView
  android:id="@+id/myTextView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Hello World, HelloWorld"
/>
```

Вот так можно получить доступ к этому элементу в коде программы:

```
TextView myTextView = (TextView)findViewById(R.id.myTextView);
```

Альтернативный способ (хотя и нерекомендуемый) — объявление макета непосредственно в коде программы (листинг 2.3).

Листинг 2.3. Объявление макета в коде программы

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  LinearLayout.LayoutParams lp;
  lp = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
                                     LayoutParams.FILL_PARENT);

  LinearLayout.LayoutParams textViewLP;
  textViewLP = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
                                             LayoutParams.WRAP_CONTENT);

  LinearLayout ll = new LinearLayout(this);
  ll.setOrientation(LinearLayout.VERTICAL);

  TextView myTextView = new TextView(this);
  myTextView.setText("Hello World, HelloWorld");

  ll.addView(myTextView, textViewLP);
  this.addContentView(ll, lp);
}
```

Все доступные в коде свойства могут быть обозначены атрибутами в XML-макете. Таким образом, обеспечение простого доступа к дизайну шаблонов и отдельных элементов пользовательского интерфейса, а также отделение кода дизайна от кода приложения делает программы более лаконичными.

В главе 4 вы научитесь создавать комплексные макеты, а также добавлять к ним представления.

Виды приложений Android

Большинство создаваемых программ относятся к одной из перечисленных категорий.

- **Программы переднего плана.** Такое приложение работает, когда оно видимо на экране, в противном случае его выполнение приостанавливается. Пример — игры или картографические мэшапы.
- **Фоновые.** Приложения, с которыми пользователи практически не взаимодействуют, за исключением их настройки. Большую часть времени они находятся в скрытом состоянии. Пример — службы экранирования звонков и SMS-автоответчики.
- **Смешанные.** Предполагают некоторую степень интерактивности, однако большую часть времени работают в фоновом режиме. Как правило, после настройки незаметны. Лишь при необходимости уведомляют пользователя о каких-либо событиях. Пример — мультимедийный проигрыватель.
- **Виджет.** Некоторые приложения представлены исключительно в виде виджетов, размещаемых на домашнем экране.

Сложные приложения трудно втиснуть в какую-то одну категорию, так как они часто содержат элементы каждого из этих типов. При создании приложения вы должны определиться с тем, как именно оно будет использоваться, и только потом приступать к его проектированию. Следует более внимательно рассмотреть некоторые конструктивные особенности каждого вида.

Программы переднего плана

Создавая приложения, которые работают на переднем плане, необходимо вдумчиво подходить к жизненному циклу `Активности` (описывается в главе 3), чтобы ее переключения в фоновый режим (и обратно) были плавными.

Приложения в Android ограничены с точки зрения контроля за своим жизненным циклом. Фоновые программы, не имеющие запущенных `Сервисов`, — главные кандидаты на закрытие с помощью системы управления

ресурсами. Это значит, что при входе в фоновый режим вы должны сохранять состояние приложения, чтобы потом иметь возможность восстановить его при выходе на передний план.

Также чрезвычайно важно, чтобы приложения, которые выводятся на экран, были удобными и интуитивно понятными. Больше о создании предсказуемых и привлекательных Активностей вы узнаете в главе 3.

Фоновые Сервисы и Широковещательные приемники

Такие приложения работают незаметно, находясь в фоне, и очень редко взаимодействуют с пользователем. Вместо того, чтобы полагаться на пользовательский ввод, они часто отслеживают сообщения или действия, посылаемые аппаратным обеспечением, системой или сторонними программами.

Вы можете создавать полностью невидимые Сервисы, но на практике все же лучше предусматривать хоть какой-то контроль со стороны пользователя. Как минимум, пользователь должен иметь возможность следить за работой Сервиса, а также при необходимости настраивать, приостанавливать или прерывать его выполнение.

Сервисы и Широковещательные приемники (Broadcast Receivers), главные составляющие фоновых приложений, подробно рассмотрены в главах 5 и 9.

Приложения смешанного вида

Часто вам потребуется создавать приложения, которые реагируют на пользовательский ввод, но не теряют работоспособности, становясь неактивными. Типичные примеры — программы для обмена текстовыми сообщениями и почтовые клиенты. Такие приложения, как правило, объединяют в себе видимые Активности и скрытые фоновые Сервисы. При взаимодействии с пользователем они должны учитывать свое состояние. Например, когда Активность отображается на экране, нужно обновлять графический интерфейс, в противном случае необходимо слать уведомления, чтобы держать пользователя в курсе происходящего. Этот процесс описывается в главе 9, а именно, в разделе об уведомлениях и Сервисах.

Виджеты

Иногда приложение может целиком состоять из единственного виджета. Используя эти компоненты (подробно описаны в главе 10), вы можете создавать интерактивные визуальные элементы, которые можно разместить на домашнем экране.

Приложения-виджеты часто применяются для отображения динамической информации, такой как заряд батареи, прогноз погоды или дата и время.

Разработка приложений для мобильных устройств

Android во многом упрощает разработку приложений для мобильных устройств, но все же важно понимать, что именно стоит за этими условностями. Существует несколько факторов, которые необходимо учитывать при написании программного обеспечения для мобильных и встраиваемых систем, в частности для Android.

ПРИМЕЧАНИЕ

В этой главе вы изучите некоторые методики и познакомитесь с рекомендациями по написанию эффективного кода для Android. В более поздних примерах, при изучении новых концепций и возможностей системы, эффективность иногда будет рассматриваться как компромисс между читаемостью кода и его лаконичностью. Предложенные примеры создавались по принципу «Делай то, что я говорю, а не то, что я делаю». Методы, которые в них используются, должны быть максимально простыми и понятными, а необязательно самыми лучшими.

Аппаратные ограничения

Миниатюрные и портативные мобильные устройства открывают перед разработчиками огромные возможности. Но при этом ограничения, связанные с размерами экрана, оперативной памятью, емкостью накопителя и мощностью процессора, порождают разные проблемы.

По сравнению с настольными ПК и ноутбуками мобильные устройства имеют:

- низкую процессорную мощность;
- ограниченную оперативную память;
- ограниченную емкость накопителей;
- малые экранные размеры и низкое разрешение;
- высокую стоимость передачи данных по Сети;
- низкую скорость и высокую латентность при передаче данных;
- ненадежное сетевое соединение;
- ограниченное время работы от батареи.

С каждым новым поколением телефонов эти ограничения постепенно нивелируются. В частности, у новых смартфонов заметно более высокое разрешение экрана, а их подключение к Сети заметно подешевело. Однако, учитывая широкий спектр доступных устройств, при проектировании рекомендуется исходить из самого худшего варианта.

Пишите эффективные программы

Производители встраиваемых и мобильных устройств, как правило, делают ставку на малые размеры и длительное время автономной работы, меньше внимания уделяя повышению мощности процессоров. Для разработчиков это выливается в потерю всех выгод, которые они получали от так называемого закона Мура (удваивание количества транзисторов в интегральных схемах каждые два года). В отличие от настольных и серверных систем, процессорная мощь которых постоянно растет, мобильные устройства стремятся к миниатюризации и повышению эффективности потребления энергии, пренебрегая значительным ростом вычислительных возможностей.

На практике это означает, что вам всегда нужно будет оптимизировать свои приложения, делая их быстрыми и отзывчивыми, не стоит и рассчитывать на то, что аппаратное обеспечение на протяжении жизненного цикла вашей программы как-то улучшится.

Поскольку эффективность кода — тема для дискуссий в области разработки программного обеспечения, я не буду здесь пытаться охватить эту тему целиком. Далее в этой главе вы получите некоторые рекомендации касательно производительности программ в Android, но сейчас просто имейте в виду, что эффективность играет чрезвычайно важную роль в системах с ограниченными ресурсами, таких как мобильные устройства.

Принимайте во внимание ограниченную емкость накопителей

Достижения в области флеш-памяти и твердотельных дисков привели к резкому увеличению объема накопителей в мобильных устройствах (хотя музыкальные коллекции в формате MP3 тоже имеют тенденцию расширяться, заполняя все доступное место). Флеш-накопители или SD-карты объемом 8Гбайт уже давно не редкость в мобильных устройствах, но оптические диски емкостью более 32Гбайт и терабайтные винчестеры пока что доступны только персональным компьютерам. Учитывая то, что большая часть встроенной памяти, как правило, используется для хранения музыки и фильмов, на большинстве устройств ваши приложения будут работать в условиях относительно ограниченного дискового пространства.

Устройства под управлением Android накладывают дополнительные ограничения, связанные с тем, что все программы должны устанавливаться на встроенную память (а не на внешние SD-карты). Из этого следует, что вам нужно следить за размером откомпилированных приложений, хотя правильное использование системных ресурсов куда более приоритетная задача.

Требуется тщательно продумывать способ хранения данных приложением. Чтобы упростить себе жизнь, можете использовать базы данных и объекты ContentProvider для хранения, повторного использования и распределения больших объемов данных (данный подход описан в главе 7).

Для данных более скромного размера, таких как настройки или информация о состоянии различных объектов, Android предоставляет специально оптимизированный для этого фреймворк, речь о котором пойдет в главе 6.

Конечно, наличие подобных механизмов не запрещает напрямую работать с файловой системой, если у вас есть такое желание или если того требуют обстоятельства. Но в таком случае вам всегда нужно думать о том, каким образом структурировать файлы, обеспечивая эффективность собственного решения.

Одна из составляющих деликатной работы с ресурсами — умение «убирать за собой». Методики наподобие кэширования полезны при ограниченном количестве сетевых запросов, но не забывайте удалять свои файлы или записи в базе данных, если они вам больше не нужны.

Проектируйте приложения для небольших экранов

Малые размеры и портативность мобильных устройств возводят препятствия на пути создания хороших интерфейсов, а пользователи все чаще требуют ярких и информационно-насыщенных решений. При создании приложений помните, что они часто только краем глаза смотрят на (небольшие) экраны своих устройств. Делайте свои программы интуитивно понятными и простыми в использовании, избавляясь от лишних элементов и размещая наиболее важную информацию в самом верху или по центру.

Графические элементы управления, наподобие тех, которые вы научитесь создавать в главе 4, — отличное средство для отображения больших объемов информации таким образом, чтобы она легко воспринималась. Вместо заполнения экрана текстом, кнопками и полями ввода, используйте для отображения данных разные цвета, геометрические фигуры и графические элементы.

Если вы планируете поддерживать сенсорные экраны (а если нет, то вам стоит этим заняться), необходимо подумать о влиянии этого типа ввода на дизайн интерфейса. Время стилусов прошло, в наши дни управление пальцами становится стандартом, и чтобы его поддерживать, ваши Представления должны иметь достаточный размер. Больше о работе с сенсорными экранами читайте в главе 15.

Сейчас Android работает на телефонах с различными размерами экрана, включая QVGA, HVGA и WVGA. По мере совершенствования дисплеев и распространения Android на рынке мобильных устройств размеры и разрешения экранов также продолжают расти. Чтобы ваше приложение хорошо выглядело и работало на всех возможных аппаратных конфигурациях, при проектировании пользовательского интерфейса важно учитывать как маленькие, так и большие экраны. С некоторыми способами оптимизации графического интерфейса для разных экранных размеров вы познакомитесь в главе 3.

Готовьтесь к низким скоростям и высокой латентности

В главе 5 вы научитесь применять в своих приложениях интернет-ресурсы. Ваши программы могут использовать все то разнообразие информации, которая доступна в Сети, благодаря чему перед вами откроются невероятные возможности.

К сожалению, мобильный Интернет часто не так быстр и надежен, как хотелось бы. При разработке своих интернет-приложений лучше исходить из того, что сетевое подключение будет медленным, непостоянным и дорогим. Ситуация меняется к лучшему благодаря безлимитным тарифам для 3G и общегородским сетям Wi-Fi, но учитывая худший вариант, вы всегда сможете поддерживать высокое качество своих приложений.

Кроме того, в результате такого подхода программы будут работоспособными в условиях потери (или полного отсутствия) сетевого подключения.

Эмулятор, входящий в состав SDK, предоставляет возможность управлять скоростью и латентностью сетевого соединения. На рис. 2.8 показана панель настроек, с помощью которой можно смоделировать условно оптимальное подключение к сети EDGE.

Рис. 2.8.

Экспериментируйте, чтобы удостовериться, что ваше приложение остается целостным и отзывчивым, несмотря на скорость, латентность и уровень доступности Сети. В некоторых ситуациях лучший выбор — ограничить функциональность программы или уменьшить количество сетевых запросов с помощью кэширования, основываясь на свойствах сетевых подключений. Подробности, как во время выполнения программы обнаруживать типы доступных соединений и определять их скорость, — в главе 13.

Цена вопроса

Если у вас есть мобильный телефон, для вас очевидно, что некоторые из его наиболее мощных функций могут стоить денег. Такие услуги, как SMS, геолокационные сервисы и передача данных по Сети иногда могут предоставляться вашим провайдером за отдельную плату.

Поэтому важно свести к минимуму денежные затраты, связанные с возможностями вашего приложения. Пользователи обязательно должны знать, что действия, которые они выполняют, могут привести к дополнительным расходам.

Рекомендуется исходить из того, что любые операции, предусматривающие взаимодействие устройства с внешним миром, платные. В некоторых случаях с помощью системных настроек пользователь может отключить потенциально платные возможности (такие как GPS или передача данных по Сети). Как разработчику вам необходимо использовать и учитывать эти настройки в своем приложении.

Для минимизации подобных издержек используйте такие рекомендации:

- передавайте как можно меньше данных;
- кэшируйте данные и результаты работы приемника GPS, чтобы устранить избыточность или дублирование сетевых запросов;
- останавливайте передачу данных и обновление показаний приемника GPS, когда ваша `Активность` не отображается на экране (если эта информация нужна только для обновления пользовательского интерфейса);
- поддерживайте минимально допустимую частоту обновлений при передаче данных и при обращении к GPS;
- планируйте ресурсоемкие обновления или передачу данных с учетом самого благоприятного времени суток, используя механизм `Сигнализации` (как показано в главе 9);
- учитывайте пользовательские настройки, касающиеся фоновой передачи данных.

Часто наиболее удачное решение — применить услуги с не самым высоким качеством, но по меньшей цене.

Используя геолокационные сервисы (описаны в главе 8), вы можете выбрать Источник данных, основываясь на его платности/бесплатности. Внутри своих приложений старайтесь предоставлять пользователю выбор между меньшей стоимостью и более высокой точностью.

В некоторых случаях стоимость услуги довольно сложно определить, или она может быть разной для разных пользователей. Затраты зависят от оператора и тарифа: одним могут предоставлять безлимитный доступ к Сети, другим — бесплатные SMS.

Вместо того чтобы всегда следовать принципу «чем дешевле, тем лучше», старайтесь давать своим пользователям возможность выбора. Например, при загрузке данных из Интернета можете спросить владельца устройства, хочет ли он использовать любые доступные соединения или же стоит ограничиться подключением по Wi-Fi.

Приложение с точки зрения пользователя

Вряд ли вы можете надеяться на то, что пользователи будут рассматривать ваше приложение в качестве самой важной особенности своего устройства.

Хотя Android уже начинает осваивать новые аппаратные платформы, основную массу устройств, на которых он установлен, все еще составляют мобильные телефоны. Помните, что большинству пользователей этих устройств в первую очередь нужен телефон и только потом они обращают внимание на SMS, электронную почту, камеру и MP3-проигрыватель. Приложения, которые вы пишите, скорее всего, они отнесут к наименее приоритетной категории — полезным мобильным утилитам.

Это не так уж и плохо: они смогут составить отличную компанию другим программам, включая Google Maps и браузер. Тем не менее, все люди разные: одни никогда не используют свои мобильные телефоны для прослушивания музыки, а другие покупают смартфоны без камеры. Но принцип многозадачности — важный и неотъемлемый фактор, благодаря которому сфера использования таких устройств расширяется.

Важно также учитывать, когда и каким образом ваши приложения будут задействованы. Люди не расстаются со своими мобильными телефонами: они пользуются ими в пути, во время пеших прогулок и даже при вождении автомобиля. Вы не можете заставить их использовать свои телефоны так, как вам этого хочется, но вы можете убедиться в том, что ваши приложения не отвлекают пользователей больше, чем это необходимо.

Что это означает с точки зрения проектирования программного обеспечения?

- **Приложение должно вести себя надлежащим образом.** Прежде всего убедитесь, что Активности приостанавливают работу, уходя в фоновый режим. Во время остановки или возобновления работы Активностей Android генерирует события, обрабатывая которые вы можете «заморозить» обновление графических элементов и отложить сетевые запросы, ведь если никто не видит ваш пользовательский интерфейс, нет никакого смысла его обновлять. Для случаев, когда работа должна продолжаться даже в фоне, в Android предусмотрен класс Service, не зависящий от графического интерфейса.

- **Приложение должно плавно переходить из фонового режима на передний план.** Учитывая многозадачность мобильных устройств, весьма вероятно, что ваши приложения будут регулярно уходить в фон и возвращаться обратно. Важно сделать так, чтобы они «возвращались к жизни» быстро и плавно. Управление процессами в Android недетерминированно: если ваше приложение находится в фоновом режиме, его работа может преждевременно завершиться для освобождения ресурсов. Все это должно быть скрыто от пользователя. Вы можете обеспечивать цельность своего приложения, сохраняя его состояние и помещая обновления в очередь — пользователь будет думать, что работа программы просто была возобновлена, не замечая повторного запуска. Переключение между состояниями должно происходить плавно, а на экран нужно выводить тот же интерфейс, который был до этого.

- **Приложение должно быть деликатным.** Ваше приложение никогда не должно перехватывать ввод данных или прерывать работу текущей Активности. Если оно не на переднем плане, следует использовать объекты Notification и Toast (подробно описаны в главе 9), вместо того, чтобы привлекать внимание пользователя прямо из окна своей программы. Существует несколько способов, с помощью которых мобильное устройство может уведомлять пользователей о разных событиях. Например, при входящем звонке телефон проигрывает мелодию, при получении сообщения мигают светодиоды, а при обнаружении новой голосовой почты в статусной строке появляется значок в виде конверта. Эти и другие методики доступны благодаря механизму уведомлений.

- **Приложение должно иметь целостный и последовательный пользовательский интерфейс.** Ваше приложение, скорее всего, будет использоваться наряду с другими программами, поэтому важно, чтобы оно имело простой и понятный графический интерфейс. Не заставляйте пользователей приспосабливаться к приложению при каждом его запуске. Работа с ним должна быть легкой, простой и очевидной,

особенно с учетом ограниченных размеров экрана и постоянных раздражителей, которые отвлекают от телефона.

- **Приложение должно быть отзывчивым.** Отзывчивость — один из наиболее важных факторов при проектировании программ для мобильных устройств. Несомненно, вы уже успели испытать разочарование от приложений, которые «тормозят» во время работы, такие ситуации еще больше раздражают, если учитывать многофункциональную природу мобильных устройств. Рискуя столкнуться с задержками, вызванными медленными и ненадежными сетевыми подключениями, необходимо использовать потоки и фоновые Сервисы, чтобы ваши Активности не теряли отзывчивости. Что еще более важно — нужно предусмотреть остановку работы своих компонентов, чтобы другие приложения тоже могли выполняться как следует.

Разработка для платформы Android

Все, что было изложено до этого момента, не имеет прямого отношения к Android. Рекомендации, о которых шла речь выше, в равной степени касаются разработки для любой мобильной платформы. В дополнение к этим общим принципам Android имеет свои особенности.

Для начала стоит потратить несколько минут на ознакомление с рекомендациями по проектированию, которые входят в состав официального руководства для разработчиков под Android. Найти их можно по адресу http://developer.android.com/guide/index.html.

Философия Android требует от ваших приложений:

- производительности;
- отзывчивости;
- целостности;
- безопасности.

Скорость и эффективность

В условиях ограниченных ресурсов скорость приложения напрямую зависит от его эффективности. Многое из того, что вы уже знаете о написании эффективного кода, применимо и к Android, но ограничения встраиваемых систем и использование виртуальной машины Dalvik не позволят воспринимать ресурсы как нечто само собой разумеющееся.

Хорошее решение — обратиться к первоисточнику. Команда разработчиков платформы Android опубликовала специфичные рекомендации по написанию эффективного кода для этой системы, поэтому вместо того, чтобы пересказывать их советы, я предлагаю посетить страницу http://developer.android.com/guide/practices/design/performance.html и ознакомиться с руководством.

ПРИМЕЧАНИЕ

Вам может показаться, что некоторые из этих рекомендаций противоречат устоявшимся методикам проектирования, например отказ от использования внутренних сеттеров и геттеров или выбор виртуальных классов вместо интерфейсов. При написании программ для систем с ограниченными ресурсами, таких как встраиваемые устройства, часто необходимо находить компромисс между общепринятыми принципами разработки и повышенными требованиями к эффективности.

Один из ключевых моментов при написании эффективного кода для Android — не нужно рассматривать встраиваемые устройства с точки зрения настольных и серверных систем.

В то время как на серверах и настольных компьютерах объем оперативной памяти 2–4 Гбайта считается обычным, типичные смартфоны «имеют на борту» около 200 Мбайт SDRAM. Память — дефицитный ресурс, и вы должны проявлять особую осторожность, чтобы использовать ее эффективно. Это означает, что необходимо думать о том, как применять очереди и кучи, ограничивать количество создаваемых объектов и следить за тем, каким образом ваши переменные влияют на использование памяти.

Отзывчивость

Отзывчивость в Android — весьма серьезный фактор. Достигается она с помощью менеджера Активностей и системы управления окнами. Если один из этих Сервисов обнаруживает неотзывчивое приложение, на экран будет выведено ужасное сообщение «Извините! Активность не отвечает» (часто в виде диалога принудительного закрытия). Такая ситуация показана на рис. 2.9.

Это сообщение модальное, оно перебирает на себя фокус и не исчезает, пока не нажмете кнопку или ваше приложение не начнет отвечать на запросы. Меньше всего вам бы хотелось показывать своему пользователю такое диалоговое окно.

При определении отзывчивости Android учитывает два условия:

- приложение должно реагировать на любые действия пользователя (нажатие клавиш или прикосновение к экрану) в течение 5 секунд;
- Широковещательный приемник должен заканчивать работу своего метода onReceive максимум за 10 секунд.

Как правило, виновники зависаний — сетевые запросы, сложные вычисления (расчет движений в игре) и файловый ввод/вывод. Существует немало методик, с помощью которых можно выполнять данные действия и держать при этом приложение в отзывчивом состоянии. В частности, речь идет о Сервисах и потоках (описаны в главе 9).



Рис. 2.9.

ПРИМЕЧАНИЕ

Диалоговое окно принудительного закрытия — крайний случай поддержания работоспособности; великодушно предоставляемые системой 5 секунд должны восприниматься как худший сценарий, но не как цель. Пользователи будут замечать постоянные задержки, если между действием и реакцией на него больше, чем полсекунды. К счастью, эффективный код, который вами уже написан, положительно влияет на отзывчивость приложений.

Безопасность

Приложения в Android имеют доступ к сети и программному обеспечению, могут распространяться самостоятельно и опираться на открытую и свободную коммуникационную платформу. Поэтому нет ничего удивительного в том, что безопасности в Android уделяется значительное внимание.

По большей части пользователи должны сами отвечать за приложения, которые они устанавливают, и за полномочия, которые они этим приложениям выдают. Модель безопасности, применяемая в Android, ограничивает доступ к некоторым услугам и возможностям, вынуждая разработчиков декларировать для своих программ полномочия, которые им необходимы. Список этих полномочий выводится пользователям непосредственно перед

установкой (больше о модели безопасности в Android в главе 15; дополнительную информацию можно получить, перейдя по ссылке http://developer.android.com/guide/appendix/faq/security.html).

Это лишает вас любых лазеек. Вам нужно думать не только о безопасности самого приложения, но и о том, чтобы его нельзя было использовать для взлома устройства. Можете применить несколько методик, которые помогают обезопасить систему (более подробно они будут рассмотрены после того, как вы изучите соответствующие технологии). В частности, необходимо:

- требовать полномочия для любых Сервисов, которые вы публикуете, или Намерений (Intents), которые транслируете;
- соблюдать особую осторожность при получении данных из внешних источников: Интернета, Bluetooth, SMS или средства мгновенного обмена сообщениями (IM);
- быть осмотрительным в тех случаях, когда ваше приложение может открыть доступ к низкоуровневому аппаратному обеспечению сторонним программам.

ПРИМЕЧАНИЕ

Для ясности и простоты многие примеры в этой книге довольно либеральны с точки зрения безопасности. При создании собственных приложений, особенно тех, которые планируете распространять, эту часть разработки не следует пускать на самотек.

Хорошее впечатление

Впечатление от использования приложения — важная часть работы, хотя оно субъективно. Что мы понимаем под хорошим впечатлением? Главная цель — чтобы приложение выглядело целостным, мгновенно запускалось и останавливалось, без заметных задержек или резких переходов.

Скорость и отзывчивость мобильного устройства не должны ухудшаться по мере его работы. Помогает в этом система управления процессами, которая, словно невидимый палач, «убивает» фоновые приложения и освобождает ресурсы, если это необходимо. Учитывая такое положение вещей, ваши программы всегда должны предоставлять целостный интерфейс, независимо от того, их работа была прервана или возобновлена.

Как правило, на устройстве под управлением Android одновременно работает несколько сторонних приложений, написанных разными разработчиками, поэтому очень важно, чтобы их взаимодействие не вызывало никаких проблем. Используя Намерения (объекты Intent), программы могут предоставлять друг другу часть функций. Осознание того, что ваше приложение может делиться своими и задействовать сторонние Активности, дает дополнительный стимул для поддержания единых принципов работы и оформления.

Создавайте свои программы целостными и интуитивно понятными, делая их удобными в использовании. Вы можете разрабатывать революционные приложения, которые ни на что не похожи, но даже в этом случае вы должны предусмотреть беспроблемную интеграцию с более объемной средой, которую предоставляет Android.

Сохраняйте данные между сессиями и, когда приложение не показывается на экране, приостанавливайте операции, расходующие процессорные мощности, сетевой трафик или емкость батареи. Если ваша программа содержит процессы, которые должны продолжаться, даже когда Активность вне поля зрения, используйте Сервисы, но скрывайте от пользователей все нюансы реализации.

Когда ваше приложение опять возвращается на передний план или повторно запускается, нужно плавно вернуться к предыдущему видимому состоянию. Находясь в фоне, любое приложение должно оставаться незаметным, но быть готовым к работе, как только это потребуется.

Вы также должны следовать рекомендациям по использованию уведомлений, применять стандартные элементы интерфейса и визуальные темы, чтобы поддерживать некую преемственность между разными приложениями.

Существует множество других методик, с помощью которых можно придать приложению плавность и удобство, и с некоторыми из них вы вскоре познакомитесь. Но перед этим (в следующих главах) узнаете больше о возможностях, которые предоставляет Android.

Приложение To-Do List

В этом примере[1] вы с нуля создадите приложение для управления списком задач, использующее стандартные графические элементы, предоставляемые Android. Основная цель данной программы — продемонстрировать базовые действия, которые необходимо выполнить для создания нового проекта.

ПРИМЕЧАНИЕ

Не переживайте, если что-либо из этого примера покажется непонятным. Некоторые возможности, используемые при создании данного приложения, включая такие классы, как ArrayAdapter, ListView и KeyListener, подробно рассмотрены в следующих главах. Позже, более тесно познакомившись с платформой Android, вы вернетесь к этому примеру, чтобы добавить в него новую функциональность.

[1] Все фрагменты кода в этом примере — часть проекта To-Do List из главы 2, их можно скачать с сайта Wrox.com.

1. Начните с создания нового проекта в Eclipse. Пройдите в меню File ▸ New ▸ Project и выберите пункт Android (как показано на рис. 2.10), затем нажмите Next.

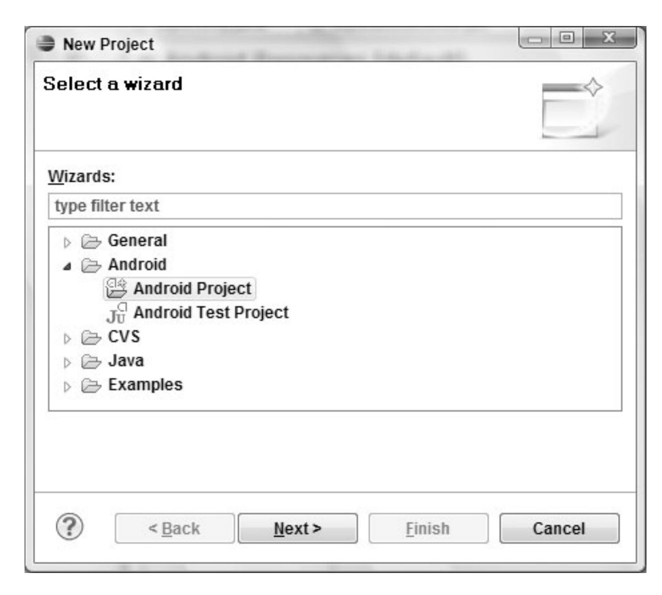


Рис. 2.10.

2. Введите подробности о своем проекте в диалоговом окне, которое появилось на экране (рис. 2.11). Application name — это имя вашего приложения (будет показываться пользователю), поле Create Activity позволяет выбрать имя для своей Активности. Закончив с вводом данных, нажмите кнопку Finish для создания нового проекта.
3. Прежде чем создавать отладочную и рабочую конфигурации, воспользуйтесь возможностью создать виртуальное устройство, на котором будет тестироваться приложение.
   3.1. Выберите пункт меню Window ▸ Android SDK and AVD Manager. В появившемся диалоге (рис 2.12) выберите на левой панели Virtual Devices и нажмите кнопку New.
   3.2. Введите имя своего устройства и выберите целевые SDK и разрешение экрана. Сделайте размер SD-карты больше, чем 8 Мбайт: укажите `12` в поле ввода (рис. 2.13).
4. Создайте отладочную и рабочую конфигурации. Чтобы это сделать, откройте меню Run ▸ Debug Configurations и Run ▸ Run Configurations, каждый раз указывая проект Todo_List и выбирая виртуальное устройство, которое вы создали в пункте 3. Вы можете оставить действие Launch Default Activity или же явно указать, что при запуске должна загружаться Активность ToDoList (рис. 2.14).

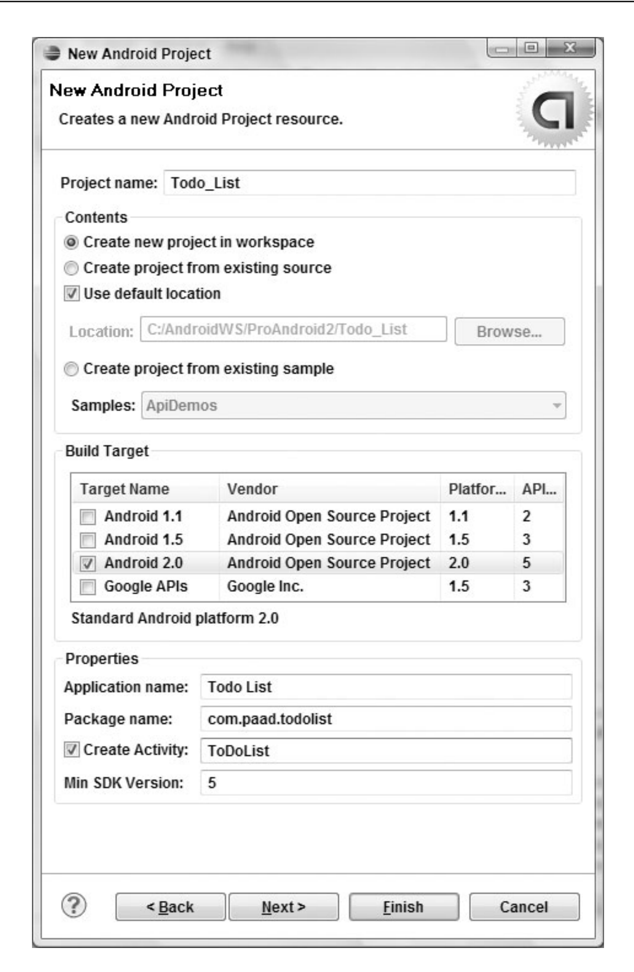


Рис. 2.11.

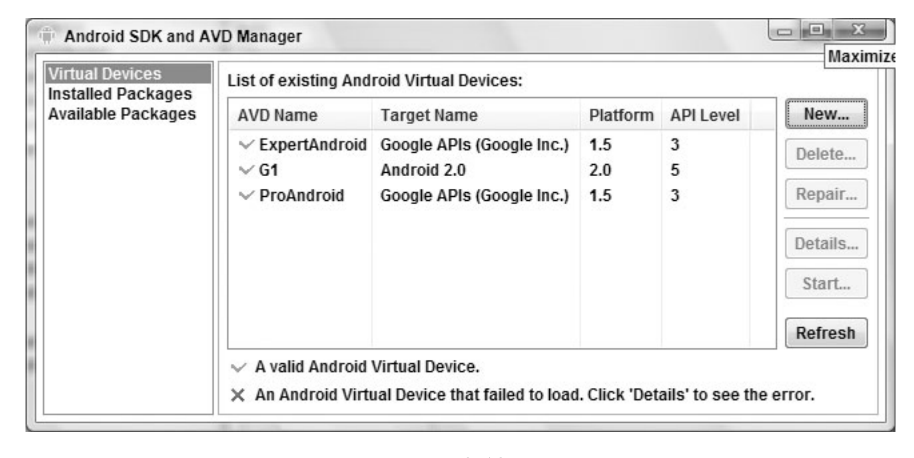


Рис. 2.12.

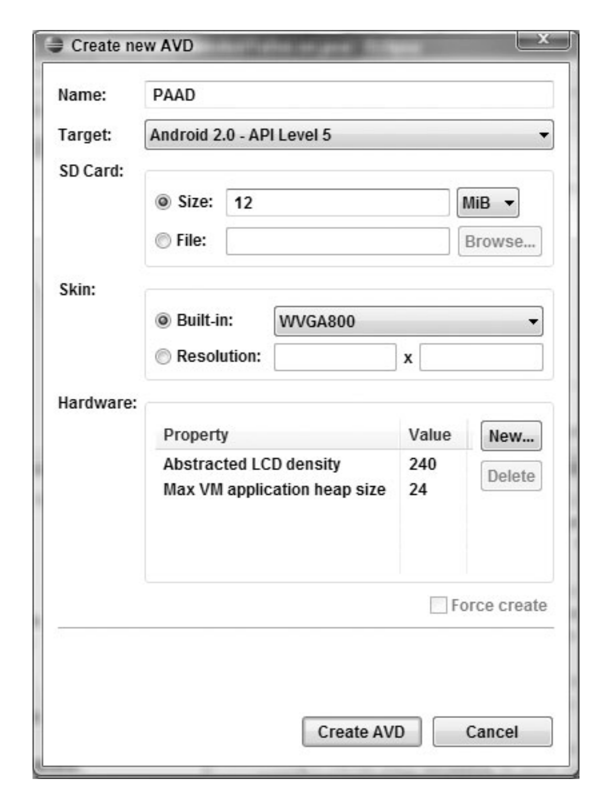


Рис. 2.13.

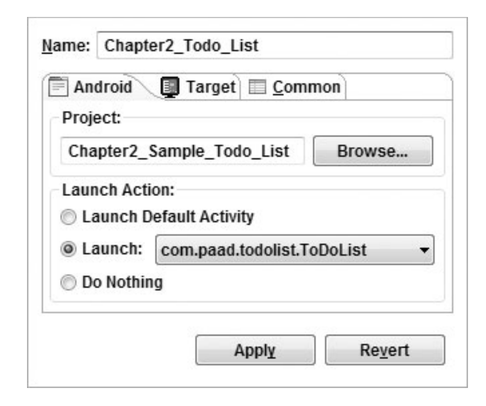


Рис. 2.14.

5. Подумайте, что именно вы хотите показать своим пользователям и какие действия те должны выполнять. Спроектируйте пользовательский интерфейс, который эти действия максимально упростит.

   В этом примере запланировано представить пользователям список задач и поле ввода для добавления новых пунктов. Оба эти элемента

доступны в библиотеках, предоставляемых платформой Android. (В главе 4 вы узнаете больше о доступных графических элементах и о том, как создавать свои собственные Представления.)

Предпочтительный способ для построения пользовательского интерфейса — описание разметки в виде файлов с ресурсами. Откройте файл main.xml из каталога res/layout, как показано на рис. 2.15.

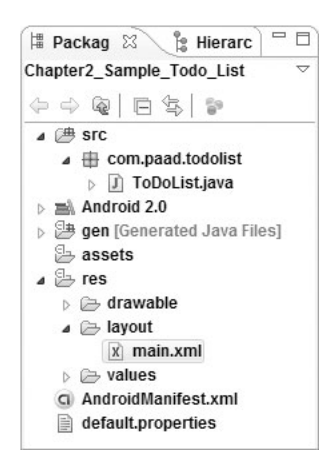


Рис. 2.15.

6. Отредактируйте главную разметку, добавив в нее элементы ListView, EditText и LinearLayout. Первые два Представления должны содержать идентификаторы, чтобы на них можно было ссылаться в коде программы.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <EditText
    android:id="@+id/myEditText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="New To Do Item"
  />
  <ListView
    android:id="@+id/myListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
</LinearLayout>
```

7. Описав пользовательский интерфейс, откройте Активность ToDoList из каталога с проектом. В этом примере все изменения, которые вы сделаете, заключаются в переопределении метода onCreate. Начните с загрузки

пользовательского интерфейса с помощью метода setContentView, затем получите ссылки на элементы ListView и EditText, используя метод findViewById.

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  // Загрузите свою разметку
  setContentView(R.layout.main);

  // Получите ссылки на элементы пользовательского интерфейса
  ListView myListView = (ListView)findViewById(R.id.myListView);
  final EditText myEditText = (EditText)findViewById(R.id.myEditText);
}
```

8. Оставаясь внутри метода onCreate, создайте строковой массив ArrayList для хранения каждого элемента списка задач. Вы можете привязать этот массив к элементу ListView с помощью нового экземпляра ArrayAdapter (класс ArrayAdapter рассматривается в главе 5).

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  ListView myListView = (ListView)findViewById(R.id.myListView);
  final EditText myEditText = (EditText)findViewById(R.id.myEditText);

  // Создайте массив для хранения списка задач
  final ArrayList<String> todoItems = new ArrayList<String>();
  // Создайте ArrayAdapter, чтобы привязать массив к ListView
  final ArrayAdapter<String> aa;
  aa = new ArrayAdapter<String>(this,
                              android.R.layout.simple_list_item_1,
                              todoItems);
  // Привяжите массив к ListView.
  myListView.setAdapter(aa);
}
```

9. В завершение сделайте так, чтобы пользователи могли добавлять в список новые задачи. Создайте для элемента EditText обработчик onKeyListener, который отслеживает нажатие центральной клавиши на манипуляторе D-pad и заносит содержимое EditText в массив, оповещая об этом объект ArrayAdapter. Затем очистите поле ввода, чтобы пользователь мог добавить следующий пункт.

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  ListView myListView = (ListView)findViewById(R.id.myListView);
  final EditText myEditText = (EditText)findViewById(R.id.myEditText);

  final ArrayList<String> todoItems = new ArrayList<String>();
```

```
    final ArrayAdapter<String> aa;
    aa = new ArrayAdapter<String>(this,
                                  android.R.layout.simple_list_item_1,
                                  todoItems);
    myListView.setAdapter(aa);

    myEditText.setOnKeyListener(new OnKeyListener() {
      public boolean onKey(View v, int keyCode, KeyEvent event) {
        if (event.getAction() == KeyEvent.ACTION_DOWN)
          if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER){
            todoItems.add(0, myEditText.getText().toString());
            aa.notifyDataSetChanged();
            myEditText.setText("");
            return true;
          }
        return false;
      }
    });
  }
```

10. Запустите рабочую или отладочную версию своего приложения — и вы увидите поле ввода, ниже которого размещен список (рис. 2.16).

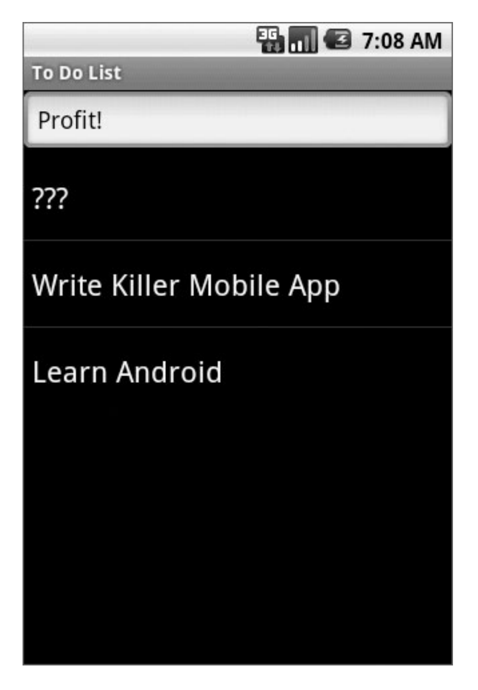


Рис. 2.16.

11. Ваше первое приложение для Android готово. Попробуйте добавить в код точки остановки, чтобы проверить отладчик и поэкспериментировать с панелью DDMS.

В нынешнем виде приложение To-Do List не особо актуально и полезно. Оно не сохраняет список задач между сессиями, вы не можете редактировать или удалять отдельные элементы, а такие свойства, как сроки выполнения и приоритеты, не записываются и даже не отображаются. В целом данный пример не соответствует большинству критериев, предъявляемых к дизайну мобильных приложений, речь о которых шла выше.

В следующих главах вы вернетесь к этому проекту и исправите некоторые его недостатки.

Средства разработки для Android

Android SDK содержит инструменты и утилиты, призванные помогать в создании, тестировании и отладке ваших проектов. Детальный обзор каждого из этих средств выходит за рамки данной книги, но краткий анализ того, что доступно, сделать все же стоит. Если вам нужна более подробная информация, ознакомьтесь с официальной документацией по адресу http://developer.android.com/guide/developing/tools/index.html.

Как уже упоминалось ранее, дополнение ADT позволяет использовать большинство из этих инструментов прямо внутри среды разработки Eclipse, а именно из панели DDMS.

- **Менеджеры виртуальных устройств и SDK.** Используются для создания и управления виртуальными устройствами в Android (AVD) и пакетами из SDK. AVD содержит эмулятор, внутри которого работает соответствующая версия Android. С его помощью можно указать поддерживаемую версию SDK, разрешение экрана, емкость доступной SD-карты и соответствующие аппаратные возможности (сенсорный экран и GPS).
- **Эмулятор.** Реализация виртуальной машины Android, предназначенная для работы внутри виртуального устройства прямо на вашем компьютере для разработки. Задействуйте эмулятор для тестирования и отладки своих приложений.
- **Служба для отслеживания процесса отладки в Dalvik (Dalvik Debug Monitoring Service или DDMS).** Используйте панель DDMS для отслеживания и контроля за виртуальными машинами Dalvik, на которых отлаживаете свои приложения.
- **Инструмент для создания пакетов (Android Asset Packaging Tool или AAPT).** Создает файлы пакетов для Android (`.apk`), готовые к распространению.
- **Android Debug Bridge (ADB).** Клиент-серверное приложение, которое предоставляет доступ к работающему эмулятору. С его помощью можно копировать файлы, устанавливать скомпилированные программные пакеты (`.apk`) и запускать консольные команды.

Вам также доступен дополнительный инструментарий.

- **SQLite3.** Задействовав данную утилиту, можно получить доступ к файлам базы данных SQLite, которые создаются и используются в Android.
- **Traceview.** Инструмент для графического анализа, с помощью которого можно просматривать трассировочные записи, принадлежащие вашему приложению.
- **MkSDCard.** Создает образ диска для SD-карты, может использоваться эмулятором для имитации внешнего накопителя.
- **Dx.** Преобразует байткод из формата Java (`.class`) в формат Android (`.dex`).
- **activityCreator.** Сценарий, создающий сборочные файлы для утилиты Ant, которые можно применять для компилирования приложений без участия ADT.
- **layoutOpt.** Инструмент, анализирующий ваши ресурсы с разметкой и предлагающий различные улучшения и оптимизации.

Рассмотрим некоторые из наиболее важных инструментов.

Менеджеры виртуальных устройств и SDK

Менеджеры AVD и SDK — инструменты, предназначенные для создания виртуальных устройств и управления ими в виде отдельных экземпляров эмулятора. Вы можете применять их как для проверки текущих, так и для установки новых доступных версий SDK.

Виртуальные устройства

Виртуальные устройства под управлением Android используются для имитации сборок программного обеспечения и аппаратных спецификаций, доступных на физических устройствах. Это дает вам возможность тестировать свои приложения на различных аппаратных платформах, без необходимости покупать настоящие телефоны.

ПРИМЕЧАНИЕ

Android SDK не содержит никаких предустановленных виртуальных устройств, поэтому, прежде чем запускать приложение в эмуляторе, необходимо создать как минимум один экземпляр AVD.

Каждое виртуальное устройство должно иметь название, целевую сборку Android (основанную на версии SDK, которую она поддерживает), емкость SD-карты и разрешение экрана. Пример такой конфигурации можно увидеть на рис. 2.17, где показано диалоговое окно для создания нового экземпляра AVD.

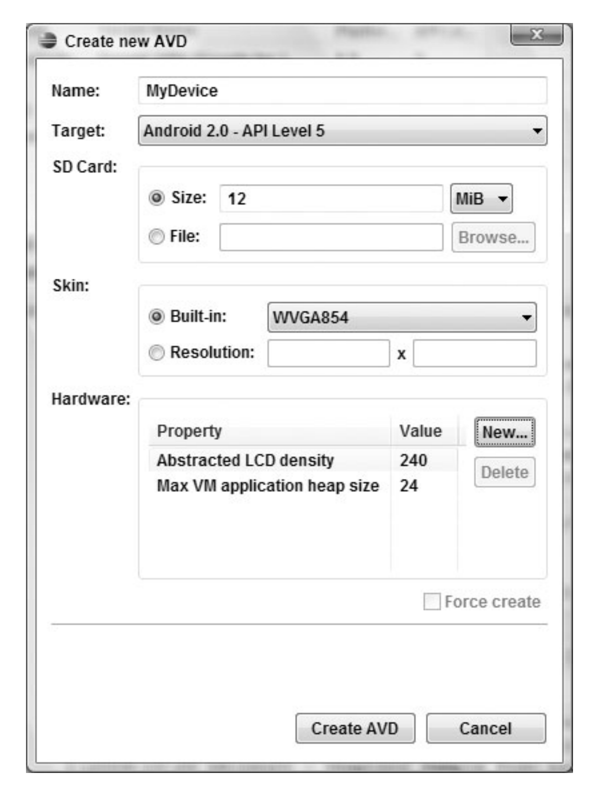


Рис. 2.17.

Кроме того, виртуальные устройства поддерживают разные аппаратные настройки и ограничения, которые можно добавить в соответствующую таблицу. В число этих параметров входят:

- максимальный размер кучи в виртуальной машине;
- плотность пикселов экрана;
- поддержка SD-карт;
- наличие манипулятора D-pad, сенсорного экрана, клавиатуры и трекбола;
- поддержка акселерометра и GPS;
- доступная память в устройстве;
- аппаратная камера и ее разрешение;
- поддержка звукозаписи.

Разные аппаратные конфигурации (в том числе разрешения экрана) представлены в виде соответствующих оболочек эмулятора. Это помогает имитировать различные типы устройств. Для достижения полного реализма

вы можете создавать собственные оболочки для каждого экземпляра AVD, сделав их похожими на устройства, которые они эмулируют.

Менеджер SDK

Используйте панели с доступными и установленными пакетами, чтобы управлять версиями SDK.

На первой панели, показанной на рис. 2.18, отображаются версии платформ SDK, документация и инструменты, которые доступны в вашей среде для разработки. При обновлении на новую версию вы можете просто нажать кнопку Update All — и менеджер обновит каждый компонент вашего SDK.

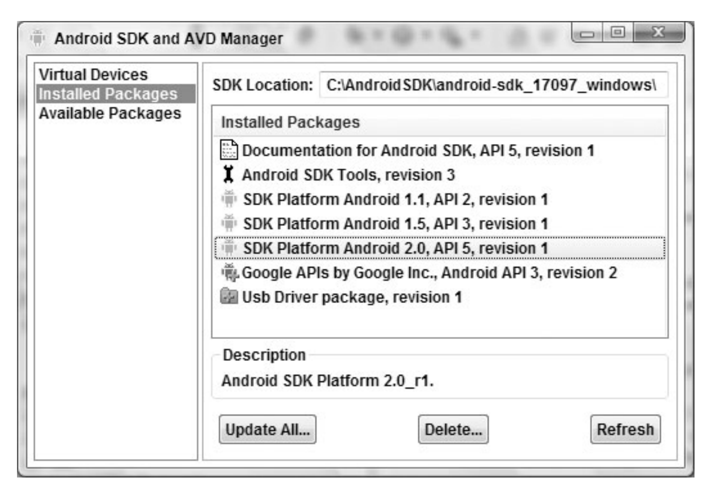


Рис. 2.18.

Вторая панель предназначена для проверки репозиториев с Android SDK на наличие доступных, но еще не установленных на вашей системе источников, пакетов и архивов. Устанавливайте флажки для выбора дополнительных пакетов, которые необходимо установить (рис. 2.19).

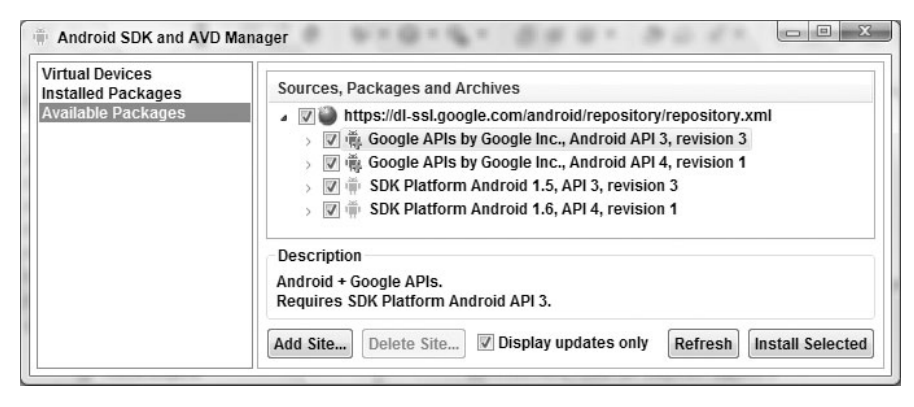


Рис. 2.19.

Эмулятор платформы Android

Эмулятор идеально подходит для тестирования и отладки ваших приложений. Это реализация виртуальной машины Dalvik, что делает его совместимым с любым телефоном на базе Android и позволяет ему запускать соответствующие приложения. Он отлично подходит для тестирования программ, потому что не привязан ни к какому конкретному аппаратному обеспечению.

Кроме доступа к полноценному сетевому подключению во время отладки программы вам предоставляется возможность настраивать скорость и латентность соединения с Интернетом. Вы также можете имитировать входящие и исходящие телефонные звонки и SMS-сообщения.

Дополнение ADT интегрирует эмулятор в Eclipse. Благодаря этому при запуске или отладке проектов он автоматически стартует внутри выбранного экземпляра AVD. Если вы не используете ADT или хотите работать с эмулятором за пределами Eclipse, то можете связываться с ним с помощью утилиты telnet и управлять из его собственной консоли. Больше подробностей об управлении эмулятором можно найти по адресу http://developer.android.com/guide/developing/tools/emulator.html.

Чтобы запустить эмулятор, сперва необходимо создать виртуальное устройство, как было описано в предыдущем разделе. Эмулятор будет загружаться внутри AVD и запускать экземпляр виртуальной машины Dalvik.

ПРИМЕЧАНИЕ

На сегодняшний день в эмуляторе пока не реализованы все аппаратные возможности, поддерживаемые платформой Android. Это касается камеры, виброзвонка, светодиодов, настоящих телефонных вызовов, акселерометра, соединений по USB, захвата аудиоданных и уровня заряда батареи.

Служба для отслеживания отладки в Dalvik (DDMS)

С помощью эмулятора вы можете оценить внешний вид своего приложения, то, как оно будет взаимодействовать с пользователем. Но чтобы увидеть, что на самом деле происходит «под капотом», нужна служба для отслеживания процесса отладки внутри Dalvik. DDMS — это мощный отладочный инструмент, который позволяет изучать активные процессы, просматривать стек и кучу, мониторить и останавливать рабочие потоки, а также исследовать файловую систему любого подключенного устройства на базе Android.

Панель DDMS в Eclipse также предоставляет упрощенный доступ к захвату изображения с экрана эмулятора и к журнальным записям, которые генерирует система LogCat.

Если вы используете дополнение ADT, DDMS полностью интегрируется в Eclipse и доступно из соответствующей панели. В противном случае можете запускать DDMS из командной строки (он автоматически подключится к любому запущенному устройству или эмулятору).

Android Debug Bridge (ADB)

Android Debug bridge (ADB) — это клиент-серверное приложение, которое позволяет подключаться к эмуляторам или устройствам, на которых запущен Android. Оно состоит из трех компонентов: фоновой службы (демона), работающей в эмуляторе, сервиса, запущенного на компьютере разработчика, и клиентской программы (наподобие DDMS), которая связывается со службой через Сервис.

ADB, как связующее звено между вашим компьютером и устройством/эмулятором, позволяет устанавливать приложения, записывать и копировать файлы, а также запускать на целевом устройстве консольные команды. Используя консоль, вы можете изменять настройки журнала и взаимодействовать с базами данных SQLite, которые хранятся на устройстве.

Дополнение ADT автоматизирует и упрощает взаимодействие с ADB, включая установку и обновление приложений, ведение журнальных записей и передачу файлов (через панель DDMS).

Чтобы узнать больше о возможностях ADB, ознакомьтесь с официальной документаций по адресу http://developer.android.com/guide/developing/tools/adb.html.

Резюме

В этой главе вы узнали, как загружать и устанавливать Android SDK, настраивать среду разработки в Eclipse на платформах Windows, Mac OS или Linux, создавать рабочие и отладочные конфигурации для своих проектов. Научились устанавливать и использовать дополнение ADT для упрощения создания новых проектов и автоматизирования цикла разработки.

Вы познакомились с некоторыми особенностями в разработке мобильных приложений. В частности, речь шла о важной роли оптимизации в обеспечении скорости и эффективности в условиях, когда увеличение времени автономной работы и уменьшение размеров устройства имеют больший приоритет, нежели мощность процессора.

Рассматривались также особенности, свойственные любым мобильным платформам, это касается разработки с учетом маленьких экранов и потенциально медленных, дорогостоящих и ненадежных сетевых соединений.

Создав приложение To-Do List, вы познакомились с виртуальными устройствами, эмулятором и средствами для разработки, с помощью которых тестируются и отлаживаются программы.

В частности, в этой главе вы:

- загрузили и установили Android SDK;
- настроили среду разработки Eclipse, а также загрузили и установили дополнение ADT;
- создали свое первое приложение и узнали, как оно работает;
- подготовили рабочие и отладочные конфигурации для своих проектов;
- узнали о разных типах приложений в Android;
- познакомились с особенностями разработки как для мобильных устройств в целом, так и для Android в частности;
- создали приложение To-Do List;
- получили представление о виртуальных устройствах, эмуляторе и инструментарии для разработки.

В следующей главе речь пойдет об Активностях и дизайне приложений. Вы изучите, как описывать программные настройки с помощью манифеста и отделять разметку пользовательского интерфейса вместе с другими ресурсами от исходных кодов. Вы также узнаете больше о жизненном цикле приложений в Android и о состояниях, в которых они могут находиться.

Глава 3
СОЗДАНИЕ ПРИЛОЖЕНИЙ И АКТИВНОСТЕЙ

Содержание главы

- Знакомство с программными компонентами Android и видами приложений, которые можно создать с их помощью.
- Жизненный цикл приложения в Android.
- Создание и комментирование программного манифеста.
- Использование внешних ресурсов для обеспечения динамической поддержки регионов, языков и аппаратных конфигураций.
- Реализация и использование своего собственного класса Application.
- Создание новых Активностей.
- Жизненный цикл Активностей и переходы между разными состояниями.

Прежде чем вы приступите к написанию собственных приложений под Android, важно понимать, как они устроены, как протекает их жизненный цикл. В этой главе вы познакомитесь с частью приложений в Android — слабосвязанными компонентами — и узнаете, как они связываются друг с другом с помощью программного манифеста. Вы также изучите, как и почему нужно использовать внешние ресурсы, после познакомитесь с компонентом Активность.

В мире программирования в последние годы наблюдается сдвиг в сторону систем, поддерживающих управляемый код, таких как JVM (Java Virtual Machine) и .NET CLR (Common Language Runtime).

В главе 1 вы узнали, что Android также использует эту модель, запуская каждое приложение в отдельном процессе, исполняющемся в собственном экземпляре виртуальной машины Dalvik. В этой главе представлено больше информации о жизненном цикле приложения и о том, как Android им управляет. Для этого необходимо изучить состояния, в которых могут находиться программные процессы — они используются при определении приоритета, что в случае нехватки ресурсов выполнение приложения будет прервано на уровне системы.

Мобильные устройства используются во всем мире и, как известно, характеризуются большим разнообразием форм и размеров. В этой главе вы узнаете, как отделять ресурсы от кода для беспроблемного запуска своих приложений на различных аппаратных платформах (в частности, на экранах с разным разрешением и плотностью пикселов) с учетом региональных особенностей и поддержкой разных языков.

Затем вы познакомитесь с классом Application и научитесь его расширять, чтобы сохранять состояние приложения.

Класс Активность, как наиболее важный программный компонент в Android, формирует основу для всех ваших экранов, содержащих пользовательский интерфейс. Вы узнаете, как создавать новые Активности, каков их жизненный цикл и как они влияют на продолжительность работы приложения.

В завершение познакомитесь с классами, наследованными от Активности, которые упрощают управление ресурсами определенным стандартным компонентам пользовательского интерфейса, таким как карты и списки.

Из чего состоят приложения в Android

Приложения в Android состоят из слабосвязанных компонентов, которые собираются воедино с помощью программного манифеста. Манифест — файл, описывающий все компоненты приложения и способы их взаимодействия, а также метаданные, в том числе требования к платформе и аппаратной конфигурации.

Компоненты, перечисленные ниже, — кирпичики, с помощью которых вы можете строить свои приложения.

- **Активности.** Уровень представления. Каждый экран приложения будет наследником класса Activity. Активности используют Представления для формирования графического пользовательского интерфейса, отображающего информацию и взаимодействующего с пользователем. С точки зрения разработки под настольные платформы Активность — эквивалент Формы (Form). Больше об этом компоненте узнаете в следующих разделах главы.
- **Сервисы.** Невидимые двигатели вашего приложения. Сервисные компоненты работают в фоновом режиме, запуская уведомления, обновляя Источники данных и видимые Активности. Используются для регулярных операций, которые должны продолжаться даже тогда, когда Активности вашего приложения не на переднем плане. О том, как создавать Сервисы, вы узнаете в главе 9.
- **Источники данных.** Хранилища информации. Данные компоненты нужны для управления базами данных в пределах одного приложения и предоставления к ним доступа извне. Источники данных задействуют-

ся при обмене информацией между разными программами. Это значит, что вы можете настраивать собственные объекты ContentProvider, открывая к ним доступ из других приложений, а также использовать чужие источники, чтобы работать с данными, которые открыли для вас внешние программы. Устройства под управлением Android содержат несколько стандартных Источников, которые предоставляют доступ к полезным базам данных, включая хранилища мультимедийных файлов и контактной информации. Создавать и использовать Источники данных вы научитесь в главе 7.

- **Намерения.** Система передачи сообщений между приложениями. Используя Намерения, вы можете транслировать сообщения на системном уровне или для конкретных Активностей или Сервисов. Тем самым диктуется необходимость выполнения заданных действий. После этого Android сам определит компоненты, которые должны обработать поступивший запрос.

- **Широковещательные приемники**. Компоненты, принимающие транслируемые Намерения. Если вы создадите и зарегистрируете объект BroadcastReceiver, ваше приложение сможет отслеживать трансляцию Намерений, которые соответствуют заданным критериям. Широковещательные приемники автоматически запустят программу, чтобы она могла ответить на принятое Намерение. Благодаря этому данный механизм идеально подходит для создания приложений, использующих событийную модель.

- **Виджеты.** Визуальные программные компоненты, которые можно добавлять на домашний экран. Этот особый вид Широковещательных приемников позволяет создавать динамические, интерактивные компоненты, которые пользователи могут встраивать в свои домашние экраны. В главе 10 вы узнаете, как создавать собственные виджеты.

- **Уведомления.** Система пользовательских уведомлений. Позволяет сигнализировать о чем-либо, не обращая на себя внимание или не прерывая работу текущей Активности. Механизм уведомлений лучше всего подходит для Сервисов и Широковещательных приемников, когда необходимо привлечь внимание пользователя. Например, принимая текстовое сообщение или входящий звонок, устройство оповещает вас, мигая светодиодами, воспроизводя звуки, отображая значки или показывая сообщения. Вы можете инициировать все эти события из собственного приложения, используя уведомления. Данный механизм рассмотрен в главе 9.

Убрав зависимость между программными компонентами, вы можете делиться и обмениваться такими самостоятельными составляющими, как Источники данных, Сервисы и даже Активности, с другими приложениями: собственными и сторонними.

Знакомство с манифестом приложения

Любое приложение, создаваемое в Android, содержит файл манифеста, AndroidManifest.xml, который хранится в корневом каталоге проекта. Манифест позволяет описывать структуру и метаданные вашего приложения, его компоненты и требования.

Манифест включает в себя узлы (теги) для каждого компонента (Активностей, Сервисов, Источников данных и Широковещательных приемников), из которых состоит ваше приложение, и с помощью Фильтров намерений (Intent Filters) и полномочий определяет, каким образом они взаимодействуют друг с другом и со сторонними программами.

В манифесте предусмотрены атрибуты для указания метаданных (значков и визуальных стилей). Надо отметить, что дополнительные узлы верхнего уровня можно использовать для описания настроек безопасности, модульных тестов (юнит-тестов), аппаратных и системных требований.

Манифест содержит корневой тег <manifest> с атрибутом package, который ссылается на пакет проекта. Как правило, этот тег также включает в себя атрибут xmlns:android, поддерживаемый системными узлами внутри файла.

Задействуйте атрибут versionCode для задания текущей версии приложения в виде целого числа. Это внутреннее значение используется для сравнения версий программы. Примените атрибут versionName для указания публичной версии, которая выводится для пользователей.

Типичный тег <manifest> показан во фрагменте кода:

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
          package="com.my_domain.my_app"
          android:versionCode="1"
          android:versionName="0.9 Beta">
          [ ... вложенные узлы манифеста ... ]
</manifest>
```

Тег <manifest> включает в себя узлы, описывающие программные компоненты, настройки безопасности, классы для тестирования и требования, из которых состоит ваше приложение. Укажем теги, доступные внутри узла <manifest>, а также фрагменты кода в формате XML, демонстрирующие, как этими тегами пользоваться.

- uses-sdk. Позволяет задать минимальную, максимальную и целевую версии SDK, которые должны быть доступны на устройстве, чтобы ваше приложение смогло правильно функционировать. Основываясь на версии SDK, которая поддерживается установленной платформой, и используя сочетание атрибутов minSDKVersion, maxSDKVersion и targetSDKVersion, вы можете ограничить круг устройств, способных запускать приложение.

Атрибут minSDKVersion указывает на минимальную версию SDK, содержащую API, которая используется в вашей программе. Если не зададите минимальную версию, применится значение по умолчанию, а ваше приложение не сможет корректно работать, если попытается получить доступ к API, которые недоступны на текущем устройстве.

Атрибут maxSDKVersion позволяет определить самую позднюю версию, которую вы готовы поддерживать. Ваше приложение будет невидимым в Android Market для устройств, управляемых системой с более свежей версией. Устанавливать значение для этого атрибута рекомендуется только в том случае, если вы абсолютно уверены, что приложение не работает на платформе с версией, выше заданной.

targetSDKVersion позволяет указать платформу, для которой вы разрабатывали и тестировали приложение. Устанавливая значение для этого атрибута, вы сообщаете системе, что для поддержки этой конкретной версии не требуется никаких изменений, связанных с прямой или обратной совместимостью:

```
<uses-sdk android:minSdkVersion="4"
          android:targetSdkVersion="5">
</uses-sdk>
```

ПРИМЕЧАНИЕ

Поддерживаемая версия SDK не совпадает с версией платформы и не может быть из нее извлечена. Например, версия платформы Android 2.0 поддерживает SDK версии 5. Чтобы получить корректную версию SDK для каждой платформы, смотрите таблицу по адресу http://developer.android.com/guide/appendix/api-levels.html.

- uses-configuration. Используйте теги uses-configuration, чтобы указать все механизмы ввода данных, поддерживаемые вашим приложением. Вы можете задать любую комбинацию, содержащую следующие устройства:
  - reqFiveWayNav — укажите для этого атрибута значение true, если вам необходимо устройство ввода, поддерживающее навигацию вверх, вниз, влево, вправо, а также нажатие выделенного элемента; в эту категорию входят как трекболы, так и манипуляторы D-pad;
  - reqHardKeyboard — если вашему приложению нужна аппаратная клавиатура, укажите значение true;
  - reqKeyboardType — позволяет задать тип клавиатуры — nokeys, qwerty, twelvekey или undefined;

- `reqNavigation` — если требуется устройство для навигации, укажите одно из следующих значений — nonav, dpad, trackball, wheel или undefined;
- `reqTouchScreen` — если вашему приложению понадобится сенсорный экран, выберите одно из следующих значений — notouch, stylus, finger или undefined.

Вы можете задать несколько поддерживаемых конфигураций, например устройство с емкостным сенсорным экраном, трекболом и аппаратной клавиатурой (либо qwerty, либо twelvekey), как показано ниже:

```
<uses-configuration android:reqTouchScreen=["finger"]
                    android:reqNavigation=["trackball"]
                    android:reqHardKeyboard=["true"]
                    android:reqKeyboardType=["qwerty"/>
<uses-configuration android:reqTouchScreen=["finger"]
                    android:reqNavigation=["trackball"]
                    android:reqHardKeyboard=["true"]
                    android:reqKeyboardType=["twelvekey"]/>
```

ВНИМАНИЕ

Определяя требуемые конфигурации, помните, что приложение не будет устанавливаться на устройствах, которые не соответствуют ни одной из заданных комбинаций. В примере, приведенном выше, устройство с qwerty-клавиатурой и манипулятором D-pad (но без сенсорного экрана или трекбола) поддерживаться не будет. В идеале вы должны разрабатывать приложения таким образом, чтобы они работали с любым сочетанием устройств ввода, в этом случае тег uses-configuration указывать необязательно.

- `uses-feature`. Одно из преимуществ Android — широкий диапазон аппаратных платформ, на которых он может работать. Используйте простые теги `uses-feature`, чтобы задать все необходимые приложению аппаратные возможности. Это предотвратит установку вашей программы на устройства, которые не соответсвуют аппаратным требованиям. Можете запросить поддержку любого необязательного для совместимых устройств оборудования. На сегодняшний день аппаратные возможности предлагают следующие варианты:
  - android.hardware.camera (если для работы приложения нужна аппаратная камера);
  - android.hardware.camera.autofocus (если требуется камера с автоматической фокусировкой).

ПРИМЕЧАНИЕ

С увеличением числа разнообразных платформ, на которых доступен Android, растет и количество дополнительного оборудования. Полный список аппаратного обеспечения для тега uses-feature можно найти по адресу http://developer.android.com/guide/topics/manifest/uses-feature-element.html.

Вы также можете использовать тег `uses-feature`, чтобы задать минимальную версию OpenGL, которая требуется для работы вашего приложения. С помощью атрибута `glEsVersion` укажите версию OpenGL ES в виде целого числа. Первые 16 бит соответствуют мажорной версии, а последние — минорной:

```
<uses-feature android:glEsVersion=" 0x00010001"
              android:name="android.hardware.camera" />
```

- `supports-screens`. После первой волны устройств с экранами HVGA в 2009 году список аппаратов под управлением Android пополнился моделями с поддержкой WVGA и QVGA. Поскольку будущие устройства, вероятно, станут оснащаться большими дисплеями, с помощью тега `supports-screen` вы можете указать экранные размеры, которые поддерживаются (и не поддерживаются) вашим приложением.

  Точные цифры будут варьироваться в зависимости от аппаратного обеспечения, но в целом соответствие размеров и разрешений экранов определяется следующим образом:

  - smallScreens — экраны с разрешением меньшим, чем обычное HVGA, как правило, речь идет о QVGA;
  - normalScreens — используется для описания экранов стандартных мобильных телефонов, как минимум HVGA, включая HVGA и WQVGA;
  - largeScreens — экраны больших размеров, значительно больше, чем у мобильного телефона;
  - anyDensity — установите значение true, если ваше приложение способно масштабироваться для отображения на экране с любым разрешением.

  В версии SDK 1.6 (API level 4) значения по умолчанию для каждого атрибута — true. Используйте этот тег для указания размеров экранов, которые вы не поддерживаете:

```
<supports-screens android:smallScreens=["false"]
                  android:normalScreens=["true"]
                  android:largeScreens=["true"]
                  android:anyDensity=["false"] />
```

ПРИМЕЧАНИЕ

По возможности нужно оптимизировать приложения для экранов с разными размерами и плотностью пикселов, используя каталоги с ресурсами, как показано далее в главе. Если вы укажете тег supports-screen, исключая определенные экранные размеры, приложение не сможет устанавливаться на устройства с неподдерживаемыми экранами.

- `application`. В манифесте может присутствовать только один экземпляр данного тега. В нем используются атрибуты, содержащие метаданные для вашего приложения (включая его название, значок и визуальный стиль). Во время разработки вы должны устанавливать атрибуту `debuggable` значение true, чтобы активизировать режим отладки, хотя для конечных версий, скорее всего, его нужно отключить.

  Тег `<application>` также играет роль контейнера, который включает в себя узлы для Активностей, Сервисов, Источников данных и Широковещательных приемников, описывающих компоненты приложения. Кроме того, вы можете задать собственную реализацию класса Application. Далее в этой главе вы узнаете, как наследовать данный класс и использовать его для управления состоянием приложения.

```
<application android:icon="@drawable/icon"
             android:theme="@style/my_theme"
             android:name="MyApplication"
             android:debuggable="true">
              [ ... вложенные теги ... ]
</application>
```

- `activity`. Тег `<activity>` требуется для каждой Активности, которую отображает приложение. Используйте атрибут `android:name` для указания имени класса Активности.

  С помощью этих тегов добавьте главную Активность, которая будет запускаться первой, а также остальные экраны и диалоговые окна, которые могут показываться. Попытка запустить Активности без соответствующего описания в манифесте приведет к выбросу исключения. Каждый тег `<activity>` поддерживает вложенные узлы `<intent-filter>`, указывающие, какие именно Намерения могут запустить Активность.

```
<activity android:name=".MyActivity" android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

- `service`. Как и в предыдущем случае, каждый класс Сервиса должен иметь тег `service` (Сервисы подробно рассматриваются в главе 9). Теги `service` поддерживают вложенные узлы `<intent-filter>`, с помощью которых происходит латентное связывание.

```
<service android:enabled="true" android:name=".MyService"></service>
```

- provider. С помощью этого тега указываются все Источники данных в приложении. Источники данных, описанные в главе 7, используются для управления доступом к базам данных и для обмена информацией в рамках одной или нескольких программ.

```
<provider android:permission="com.paad.MY_PERMISSION"
          android:name=".MyContentProvider"
          android:enabled="true"
          android:authorities="com.paad.myapp.MyContentProvider">
</provider>
```

- receiver. Добавляя в манифест тег receiver, можно зарегистрировать Широковещательный приемник, не запуская при этом приложение. Как убедитесь в главе 5, Широковещательные приемники отслеживают события на глобальном уровне: пройдя регистрацию, они начнут срабатывать при трансляции системой или приложением соответствующего Намерения. Регистрируя их в манифесте, можете сделать этот процесс полностью анонимным. При трансляции соответствующего Намерения ваше приложение стартует автоматически, запуская зарегистрированный Приемник.

```
<receiver android:enabled="true"
          android:label="My Intent Receiver"
          android:name=".MyIntentReceiver">
</receiver>
```

- uses-permission. Теги uses-permission как часть системы безопасности описывают полномочия, которые, по вашему мнению, нужны приложению для полноценной работы. Добавленные полномочия предоставляются пользователю до установки. Для использования многих стандартных сервисов в Android требуются полномочия (в частности, для действий, связанных с платными услугами и безопасностью, таких как телефонные звонки, прием SMS или использование геолокационных сервисов).

```
<uses-permission android:name="android.permission.ACCESS_LOCATION"/>
```

- permission. Сторонние приложения также могут указывать полномочия, прежде чем предоставлять доступ к общим программным компонентам. Чтобы ограничить доступ к компоненту приложения, вы должны описать соответствующие полномочия в манифесте. Для этого необходимо использовать тег permission.

  Компоненты текущего приложения могут требовать полномочия с помощью атрибутов android:permission. Другие программы должны содержать в своем манифесте теги uses-permission, чтобы использовать эти защищенные компоненты.

  Внутри тега permission вы можете указать уровень доступа, который обеспечивается данным полномочием (normal, dangerous, signature, signatureOrSystem), метку и внешний ресурс, содержащий описание

и объяснение рисков, которыми сопровождается выдача этого полномочия.

```
<permission android:name="com.paad.DETONATE_DEVICE"
            android:protectionLevel="dangerous"
            android:label="Self Destruct"
            android:description="@string/detonate_description">
</permission>
```

- `instrumentation`. Классы, производные от Instrumentation, предоставляют фреймворк для тестирования программных компонентов во время их выполнения. Они содержат методы-перехватчики, с помощью которых отслеживаются работа программы и ее взаимодействия с системными ресурсами.

```
<instrumentation android:label="My Test"
                 android:name=".MyTestClass"
                 android:targetPackage="com.paad.aPackage">
</instrumentation>
```

Подробное описание манифеста и всех этих тегов можно найти по адресу http://developer.android.com/guide/topics/manifest/manifest-intro.html.

Мастер создания проектов в составе ADT (New Project Wizard) автоматически добавляет файл с манифестом для каждого нового проекта.

Вы еще вернетесь к манифесту, как только познакомитесь со всеми компонентами, из которых состоит приложение.

Использование редактора манифеста

Дополнение ADT включает визуальный редактор манифеста, поэтому можно обойтись без ручного редактирования исходного XML-кода.

Чтобы использовать редактор манифеста в Eclipse, щелкните правой кнопкой мыши по файлу `AndroidManifest.xml` в каталоге своего проекта и выберите пункт Open With ▸ Android Manifest Editor. На экране появится панель с описанием манифеста, как показано на рис. 3.1. В ней содержится высокоуровневое представление структуры вашего приложения, с его помощью вы можете изменять информацию о версии программы и корневые узлы манифеста, включая `<uses-sdk>` и `<uses-features>`, описанные ранее в этой главе. Здесь находятся ссылки для визуального представления разделов Application, Permissions, Instrumentation, а также ярлык для перехода к редактированию исходного кода в формате XML.

Каждая из следующих вкладок содержит визуальный интерфейс для управления настройками приложения, безопасности и тестирования, а последняя (использующая имя файла с манифестом) открывает доступ к исходному XML-коду.

Android Manifest

Manifest General Attributes

Defines general information about the AndroidManifest.xml

Package	com.paad.todolist	Browse...
Version code	1	
Version name	1.0	Browse...
Shared user id		Browse...
Shared user label		Browse...

Manifest Extras

Uses Sdk

Add... Remove... Up Down

Attributes for Uses Sdk

The uses-sdk tag describes the SDK features that the containing package must be running on to operate correctly.

Min SDK version	4	Browse...
Target SDK version	4	Browse...
Max SDK version		

Exporting

To export the application for distribution, you have the following options:

- Use the Export Wizard to export and sign an APK
- Export an unsigned APK and sign it manually

Links

The content of the Android Manifest is made up of three sections. You can also edit the XML directly.

Application: Activities, intent filters, providers, services and receivers.

Permission: Permissions defined and permissions used.

Manifest | Application | Permissions | Instrumentation | AndroidManifest.xml

Рис. 3.1.

Особый интерес представляет вкладка Application, показанная на рис. 3.2. Используйте ее для управления узлом `application` и деревом компонентов приложения.

Android Manifest Application

Application Toggle

The application tag describes application-level components contained in the package, as well as general application attributes.

Define an <application> tag in the AndroidManifest.xml

Application Attributes

Defines the attributes specific to the application.

Name		Persistent	
Theme		Enabled	
Label		Debuggable	
Icon	@drawable/icon	Manage space activity	
Description		Allow clear user data	
Permission		Test only	
Process		Backup agent	
Task affinity		Allow backup	
Allow task reparenting		Kill after restore	
Has code		Restore needs application	

Application Nodes

.ToDoList (Activity)
Intent Filter
android.intent.action.MAIN (Action)
android.intent.category.LAUNCHER (Category)

Add... Remove... Up Down

Manifest | Application | Permissions | Instrumentation | AndroidManifest.xml

Рис. 3.2.

На панели Application Attributes укажите свойства приложения — значок, метку и визуальный стиль. Ниже находится дерево Application Nodes, с помощью которого можно управлять программными компонентами, в том числе их атрибутами и любыми вложенными Фильтрами намерений, связанными с ними.

Жизненный цикл приложения в Android

В отличие от большинства традиционных платформ в Android приложения имеют ограниченный контроль над жизненным циклом. Программные компоненты должны отслеживать изменения в состоянии приложения и реагировать на них соответствующим образом, уделяя особое внимание подготовке к преждевременному завершению работы.

По умолчанию каждое приложение в Android работает в собственном процессе — отдельном экземпляре виртуальной машины Dalvik. Управление памятью и процессами — исключительно прерогатива системы.

ПРИМЕЧАНИЕ

Хоть это и большая редкость, но все же можно сделать так, чтобы программные компоненты одного приложения работали в разных процессах или чтобы несколько приложений использовали один и тот же процесс. Для это нужно установить атрибут android:process для тега, который описывает соответствующий компонент внутри манифеста.

Android активно управляет своими ресурсами, делая все возможное, чтобы устройство оставалось отзывчивым. То есть работа процессов (вместе с приложениями, которые они в себе выполняют) в некоторых случаях может быть завершена без предупреждения. Это касается ситуаций, когда необходимо выделить ресурсы для приложений с более высоким приоритетом, которые, как правило, должны в этот момент взаимодействовать с пользователем. Назначение приоритетов для процессов рассматривается в следующем разделе.

Приоритеты приложений и состояния процессов

Порядок, в котором завершается работа процессов с целью освобождения ресурсов, определяется их приоритетами. Этот показатель берется из самого приоритетного компонента.

Если приоритет двух приложений одинаковый, первым будет закрыто то, которое дольше всего проработало с пониженным приоритетом. На приоритет процесса также влияют межпрограммные связи. Допустим, одно

приложение зависит от Сервиса или Источника данных, которые предоставляются другим приложением. Из этого следует, что у второго приложения приоритет как минимум не ниже, чем у первого.

ПРИМЕЧАНИЕ

Любые приложения в Android продолжают работу и остаются в памяти до тех пор, пока системе не потребуются ресурсы для других программ.

На рис. 3.3 показана иерархия приоритетов, определяющая порядок, в котором прерывается работа приложений.

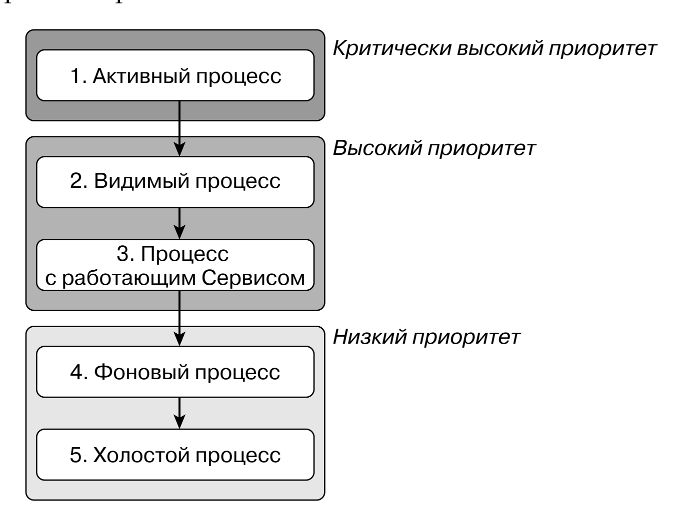


Рис. 3.3.

Важно, чтобы структура приложения была корректной и чтобы его приоритет соответствовал работе, которую оно выполняет. В противном случае приложение может закрыться во время выполнения каких-то важных действий.

Опишем каждое из программных состояний, показанных на рис. 3.3, и объясним, как их определять, учитывая компоненты приложения.

- **Активные процессы.** Активные процессы (на переднем плане) содержат компоненты, взаимодействующие с пользователем. Android поддерживает их отзывчивость, освобождая дополнительные ресурсы. Как правило, таких процессов очень мало и они закрываются в самую последнюю очередь.

  Активные процессы включают в себя:

  - объекты Activity в активном состоянии, то есть те, которые находятся на переднем плане и отвечают на пользовательские события (более подробно о состоянии Активностей далее в этой главе);

- Широковещательные приемники, обрабатывающие события с помощью методов onReceive;
- Сервисы, в которых запущены обработчики onStart, onCreate или onDestroy;
- Сервисы, предназначенные для работы на переднем плане.

• **Видимые процессы.** Видимые, но не выполняющиеся в данный момент процессы содержат Активности, которые отображаются на экране. Речь о видимых Активностях, которые не находятся на переднем плане и не отвечают на пользовательские события. Это происходит, когда Активность частично перекрыта (диалоговым окном или другой полупрозрачной Активностью). Процессов, которые выводятся на экран, очень мало, поэтому их работа прерывается только в крайнем случае, если не хватает ресурсов для активных приложений.

• **Процессы с запущенными Сервисами.** Это процессы, содержащие работающие Сервисы. Компоненты Service могут выполняться непрерывно и не должны иметь графического интерфейса. Поскольку фоновые Сервисы не взаимодействуют с пользователем напрямую, они получают немного меньший приоритет, чем видимые Активности. Наличие таких Сервисов выводит процесс на передний план и делает его преждевременное завершение возможным, только если потребуются ресурсы для активных или видимых приложений. Больше о Сервисах читайте в главе 9.

• **Фоновые процессы.** Процессы, не имеющие ни видимых Активностей, ни работающих Сервисов. Как правило, существует множество фоновых процессов, работа которых будет завершаться по принципу «последний запущенный закрывается первым», чтобы освободить ресурсы для приложений, работающих на переднем плане.

• **Холостые процессы.** Для улучшения общей производительности системы Android часто сохраняет в памяти приложения, которые завершили жизненный цикл. Android поддерживает этот кэш, чтобы уменьшить время повторного запуска программ. Работа таких процессов прерывается при необходимости.

Отделение ресурсов от кода программы

Неважно, для какой системы ведется разработка, такие ресурсы, как изображения и строковые константы, всегда лучше держать за пределами исходного кода. Android поддерживает выделение ресурсов во внешние файлы, начиная с простых значений: строк и цветов, заканчивая более сложными данными вроде изображений (объектов Drawable), анимации и визуальных стилей. Возможно, наиболее мощные ресурсы, которые можно отделить от кода программы, — менеджеры компоновки.

Внешние ресурсы легче поддерживать, обновлять и контролировать. Вы также можете описывать альтернативные ресурсы для поддержки различного аппаратного обеспечения и локализации.

В этом разделе вы увидите, как Android динамически выбирает данные из дерева ресурсов, содержащего разные значения для разных аппаратных конфигураций, языков и регионов. Это позволит описывать уникальные значения для конкретных языков, стран, экранов и клавиатур. При запуске приложения Android автоматически выберет ресурс с соответствующими данными, не требуя ни единой строчки кода.

Кроме всего прочего это дает вам возможность изменять разметку, учитывая размер экрана и его ориентацию, выводить разные текстовые подсказки в зависимости от языка и страны.

Создание ресурсов

Ресурсы приложения хранятся в каталоге res/ внутри дерева вашего проекта. Каждый тип ресурсов представлен в виде подкаталога, содержащего соответствующие данные.

При создании нового проекта дополнение ADT автоматически добавит в него каталог res с подкаталогами values, drawable-ldpi, drawable-mdpi, drawable-hdpi и layout. В них будут храниться следующие ресурсы: разметка по умолчанию, значок приложения и определения строковых констант (рис. 3.4).



Рис. 3.4.

Обратите внимание, что каждый из каталогов drawable-* содержит разные значки — для экранов с низким, средним и высоким значением DPI.

Разные каталоги предусмотрены для девяти главных типов ресурсов: простых значений, ресурсов Drawable, менеджеров компоновки, анимации, стилей, меню, настроек поиска, XML и «сырых» (необработанных) данных. При сборке приложения эти ресурсы скомпилируются самым эффективным образом и включатся в программный пакет.

При этом генерируется файл для класса R, содержащий ссылки на все ресурсы проекта, что позволяет ссылаться на ресурсы внутри кода программы. Это дает одно преимущество — проверку синтаксиса во время разработки.

Следующие разделы описывают виды ресурсов, соответствующие приведенным выше категориям, а также информацию, как создавать ресурсы для приложений.

Имена файлов для ресурсов должны состоять исключительно из букв в нижнем регистре, чисел, а также символов . (точка) и _ (нижнее подчеркивание).

Создание простых значений

Поддерживаются простые значения — строки, цвета, размеры и массивы (строковые и целочисленные), эти данные хранятся в формате XML внутри каталога res/values.

Используя теги, указывайте типы хранимых значений, как показано в листинге 3.1 на примере простого XML-файла.

Листинг 3.1. Простые значения в формате XML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">To Do List</string>
  <color name="app_background">#FF0000FF</color>
  <dimen name="default_border">5px</dimen>
  <array name="string_array">
    <item>Item 1</item>
    <item>Item 2</item>
    <item>Item 3</item>
  </array>
  <array name="integer_array">
    <item>3</item>
    <item>2</item>
    <item>1</item>
  </array>
</resources>
```

В этом примере содержатся все доступные типы простых значений. Каждый тип ресурсов принято хранить в отдельном файле, например файл `res/values/strings.xml` включает только строковые константы.

В следующих разделах подробно рассматривается процесс описания простых ресурсов.

Строки

Внешние строковые ресурсы помогают поддерживать совместимость внутри вашего приложения, упрощая процесс создания локализованных версий.

Строковые ресурсы обозначаются тегом <string>, как показано в следующем фрагменте:

```
<string name="stop_message">Stop.</string>
```

Android поддерживает простую текстовую разметку, поэтому вы можете использовать теги <b>, <i> и <u> из языка HTML, чтобы выделять части текста полужирным, наклонным или подчеркнутым стилем соответственно:

```
<string name="stop_message"><b>Stop.</b></string>
```

Можно использовать строковые ресурсы в качестве входящих параметров для метода String.format. Однако String.format не поддерживает стилизацию текста, описанную выше. Чтобы форматировать строку, при создании ресурса экранируйте HTML-теги:

```
<string name="stop_message">&lt;b>Stop&lt;/b>. %1$s</string>
```

Внутри кода используйте метод Html.fromHtml, чтобы преобразовать эту строку обратно в форматированную последовательность символов:

```
String rString = getString(R.string.stop_message);
String fString = String.format(rString, "Collaborate and listen.");
CharSequence styledString = Html.fromHtml(fString);
```

Цвета

Для описания цветовых ресурсов применяется тег <color>. Указывайте цвет с помощью символа #, за которым следуют альфа-канал (необязательно) и значения для красного, зеленого и синего цветов в виде одного или двух шестнадцатеричных чисел. Поддерживаются следующие форматы записи:

- #RGB;
- #RRGGBB;
- #ARGB;
- #AARRGGBB.

В примере описываются два цвета: полностью непрозрачный синий и полупрозрачный зеленый:

```
<color name="opaque_blue">#00F</color>
<color name="transparent_green">#7700FF00</color>
```

Размеры

Ссылки на размеры чаще всего встречаются внутри ресурсов со стилями и разметкой. Они пригодятся при создании констант — толщины рамки или высоты шрифта.

Чтобы описать ресурс, используйте тег <dimen>, указывая масштаб и один из видов размерности:

- px (экранные пикселы);
- in (физические дюймы);
- pt (физические точки);
- mm (физические миллиметры);
- dp (аппаратно-независимые пикселы, которые вычисляются относительно экрана с плотностью 160 dpi);
- sp (пикселы, не зависящие от масштаба).

В итоге можно описывать размеры не только в абсолютных, но и в относительных значениях, которые зависят от разрешения и плотности экрана, упрощая тем самым масштабирование интерфейса на разных устройствах.

В следующем примере указаны размеры для большого шрифта и стандартной рамки:

```
<dimen name="standard_border">5dp</dimen>
<dimen name="large_font_size">16sp</dimen>
```

Визуальные стили и темы

Ресурсы со стилями позволяют поддерживать единство внешнего вида приложения с помощью атрибутов, используемых Представлениями. Чаще всего визуальные стили и темы используются для хранения цветовых значений и шрифтов для программы.

Вы можете легко менять внешний вид приложения, указывая различные стили в качестве темы в манифесте своего проекта.

Чтобы создать стиль, используйте тег <style>, включающий атрибут name, а также один или несколько вложенных узлов item. Каждый тег item, в свою очередь, также должен иметь атрибут name, содержащий тип описываемого значения (например, размер шрифта или цвет). Внутри тега должно храниться само значение. Пример описания визуального стиля показан в следующем фрагменте:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="StyleName">
    <item name="attributeName">value</item>
  </style>
</resources>
```

Тег <style> поддерживает наследование с помощью атрибута parent, благодаря чему стили можно свободно варьировать.

В следующем примере демонстрируются два стиля (могут быть использованы в качестве визуальной темы): базовый, описывающий несколько свойств для текста, и дочерний, изменяющий ранее заданный шрифт на более мелкий:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="BaseText">
    <item name="android:textSize">14sp</item>
    <item name="android:textColor">#111</item>
  </style>
  <style name="SmallText" parent="BaseText">
    <item name="android:textSize">8sp</item>
  </style>
</resources>
```

Ресурсы Drawable

Ресурсы Drawable содержат растровые и растягиваемые (NinePatch) изображения. В эту категорию также входят сложные композитные ресурсы, такие как LevelListDrawables и StateListDrawables, которые могут быть описаны в формате XML.

Композитные ресурсы и изображения NinePatch более подробно рассматриваются в следующей главе.

Все ресурсы Drawable хранятся в виде отдельных файлов в каталоге res/drawable. Идентификаторами для них служат имена файлов в нижнем регистре без расширения.

ПРИМЕЧАНИЕ

Предпочтительный формат для растровых ресурсов — PNG, но JPEG и GIF также поддерживаются.

Разметка

Ресурсы с разметкой (или менеджеры компоновки) позволяют отделять уровень представления от бизнес-логики. С помощью разметки вы можете проектировать пользовательские интерфейсы в формате XML, вместо того чтобы описывать их в коде программы.

Чаще всего разметка применяется при описании пользовательского интерфейса для Активности. Создав разметку в формате XML, можно загрузить ее в Активность с помощью метода setContentView (как правило, внутри обработчика onCreate). Вы также можете получать ссылки на экземпляры разметки, содержащиеся в других ресурсах (например, разметка для каждой строки в элементе ListView). Более подробная информация об использовании и создании разметки внутри Активностей — в главе 4.

Использование менеджеров компоновки — рекомендуемый подход при проектировании пользовательских интерфейсов в Android. Отделяя разметку от кода программы, вы получаете возможность оптимизировать пользовательский интерфейс для различных аппаратных конфигураций, учитывая размеры экрана, ориентацию, наличие клавиатуры и сенсорного экрана.

Каждый ресурс, описывающий разметку, хранится в отдельном файле в каталоге res/layout. Имя файла выступает как идентификатор ресурса.

Исчерпывающее описание менеджеров компоновки и элементов интерфейса содержится в следующей главе, но в качестве примера в листинге 3.2 представлен ресурс, добавленный мастером создания проектов. Он содержит разметку LinearLayout (описана в главе 4), которая выступает в качестве контейнера для элемента TextView, отображающего приветствие Hello World!.

Листинг 3.2. Разметка с приветствием Hello World!

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World!"
  />
</LinearLayout>
```

Анимация

Android поддерживает два вида анимации. Анимация, основанная на расчете промежуточных кадров, используется для поворачивания, перемещения, растягивания и затемнения Представлений. Вы также можете создавать пошаговую анимацию, чтобы последовательно выводить на экран изображения. Полный обзор создания, использования и применения анимации приводится в главе 15.

Описывая анимацию в виде внешнего ресурса, можно использовать одну и ту же последовательность в разных местах программы. Вы также получаете возможность выбирать анимацию в зависимости от аппаратных особенностей устройства или ориентации в пространстве.

Анимация, основанная на расчете промежуточных кадров. Каждый экземпляр анимации данного типа хранится в отдельном XML-файле внутри каталога res/anim. Аналогично разметке и объектам Drawable имена файлов с описанием анимации служат идентификаторами для ресурсов.

Анимацию можно задать в виде изменений параметров alpha (затемнение), scale (масштабирование), translate (перемещение) или rotate (поворот).

В табл. 3.1 приведены допустимые атрибуты и значения, которые поддерживаются при описании анимации.

Таблица 3.1. Атрибуты для описания анимации

Тип анимации	Атрибуты	Допустимые значения
Alpha	fromAlpha/toAlpha	Значения типа Float от 0 до 1
Scale	fromXScale/toXScale	Значения типа Float от 0 до 1
	fromYScale/toYScale	Значения типа Float от 0 до 1
	pivotX/pivotY	Описывает центральную точку масштабирования в процентах от ширины/высоты — от 0 % до 100 %
Translate	fromX/toY	Значения типа Float от 0 до 1
	fromY/toY	Значения типа Float от 0 до 1
Rotate	fromDegrees/toDegrees	Значения типа Float от 0 до 360 °
	pivotX/pivotY	Описывает опорную точку для поворота в процентах от ширины/высоты — от 0 % до 100 %

Вы можете комбинировать разные экземпляры анимации, используя тег `set`. Такой набор содержит одно или несколько анимационных преобразований и поддерживает различные дополнительные теги и атрибуты, с помощью которых можно указать, как и когда должен запускаться каждый экземпляр.

Перечислим некоторые атрибуты, доступные для тега `set`:

- `duration` — продолжительность анимации в миллисекундах;
- `startOffset` — миллисекундная задержка перед началом анимации;
- `fillBefore` — установите значение true, чтобы применить преобразование перед началом анимации;
- `fillAfter` — установите значение true, чтобы применить преобразование после завершения анимации;
- `interpolator` — описывает изменения в скорости эффекта.

Доступные виды интерполяции рассматриваются в главе 15. Чтобы использовать один из них, укажите ссылку на системный ресурс с анимацией вида `android:anim/interpolatorName`.

ПРИМЕЧАНИЕ

Если не использовать атрибут startOffset, все анимационные эффекты внутри набора воспроизводятся одновременно.

В следующем примере показан анимационный набор, с помощью которого целевой элемент одновременно поворачивается на 360°, сжимается и исчезает:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <rotate
    android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="500"
    android:duration="1000" />
  <scale
    android:fromXScale="1.0"
    android:toXScale="0.0"
    android:fromYScale="1.0"
    android:toYScale="0.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="500"
    android:duration="500" />
  <alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:startOffset="500"
    android:duration="500" />
</set>
```

Пошаговая анимация. Она подразумевает создание последовательности объектов Drawable, каждый из которых будет отображаться в качестве фона для Представления на протяжении указанного промежутка времени.

В отличие от анимации, основанной на расчете промежуточных кадров, пошаговая хранится в виде ресурсов Drawable в каталоге res/drawable. Имена файлов (без расширения `.xml`) используются в качестве идентификаторов.

В следующем фрагменте представлена простая анимация, в ходе которой происходит перебор последовательности растровых ресурсов и вывод каждого из них на экран (отображаются полсекунды). Чтобы использовать этот пример, необходимо создать три новых ресурса с изображениями rocket (1, 2, 3).

```
<animation-list
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:oneshot="false">
  <item android:drawable="@drawable/rocket1" android:duration="500" />
  <item android:drawable="@drawable/rocket2" android:duration="500" />
  <item android:drawable="@drawable/rocket3" android:duration="500" />
</animation-list>
```

Меню

Описывайте меню в виде отдельных ресурсов в формате XML, вместо того чтобы создавать их прямо в коде приложения. Это поспособствует

дальнейшему повышению автономности уровня представления вашей программы.

Данный вид ресурсов может использоваться для описания как главного (принадлежащего Активности), так и контекстного меню, и предоставляет те же свойства, какие доступны при создании меню программным способом. Меню, описанное в формате XML, загружается в виде программного объекта с помощью метода inflate, принадлежащего Сервису MenuInflator. Как правило, это происходит внутри обработчика onCreateOptionsMenu. Больше подробностей смотрите в главе 4.

Описание каждого экземпляра меню хранится в отдельном файле в каталоге res/menu. Имена файлов автоматически становятся идентификаторами ресурсов. Именно такой способ определения меню наиболее предпочтителен в Android.

Исчерпывающее описание свойств меню содержится в следующей главе. В листинге 3.3 представлен простой пример.

Листинг 3.3. Описание простого ресурса с меню

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_refresh"
        android:title="Refresh" />
  <item android:id="@+id/menu_settings"
        android:title="Settings" />
</menu>
```

Использование ресурсов

Кроме ресурсов, созданных собственноручно, вы можете задействовать в своих приложениях и некоторые системные ресурсы, предоставляемые платформой Android. Их можно загрузить напрямую в коде программы или указать в виде ссылки внутри других ресурсов (например, описание разметки ссылается на ресурс с описанием размеров).

Далее в этой главе вы научитесь описывать альтернативные ресурсы для разных языков, регионов и аппаратных конфигураций. Важно отметить, что можно выбрать конкретную версию ресурсов. Android автоматически определяет наиболее подходящие значения для заданного идентификатора, основываясь на текущем аппаратном обеспечении и установках устройства.

Применение ресурсов в коде программы

Доступ к ресурсам из кода программы происходит с помощью статического класса R. Он генерируется на основе ваших внешних ресурсов, создается во время компиляции проекта, содержит статические подклассы

для всех типов ресурсов, для которых был описан хотя бы один экземпляр. Например, стандартный новый проект включает подклассы R.string и R.drawable.

ВНИМАНИЕ

Если вы используете дополнение ADT в Eclipse, класс R создастся автоматически при любых изменениях в файлах или каталогах с ресурсами. В ином случае при компилировании проекта вам понадобится утилита AAPT. Класс R генерируется автоматически; не пытайтесь править его вручную, потому как все изменения будут утеряны при повторной генерации.

Каждый из подклассов внутри класса R предоставляет доступ к соответствующим ресурсам с помощью свойств, имена которых совпадают с идентификаторами, например R.string.app_name или R.drawable.icon. Значения этих свойств ссылаются на соответствующие пути в таблице ресурсов, а не на их экземпляры.

Если какой-нибудь из конструкторов или любой другой метод (например, setContentView) принимает в качестве параметра идентификатор ресурса, можете передать ему одно из вышеописанных свойств, как показано в следующем фрагменте кода:

```
// Загрузка ресурса.
setContentView(R.layout.main);
// Отображение всплывающего диалогового окна, которое
// выводит строковой ресурс в качестве сообщения об ошибке.
Toast.makeText(this, R.string.app_error, Toast.LENGTH_LONG).show();
```

Если вам нужен непосредственно экземпляр ресурса, используйте вспомогательные методы, чтобы извлечь его из таблицы. Таблица ресурсов представлена в виде объекта класса Resources.

Поскольку данные методы ведут поиск в таблице ресурсов, принадлежащей приложению, они не могут быть статическими. Применяйте метод getResources из контекста своего приложения для доступа к экземпляру класса Resources, как показано в следующем фрагменте:

```
Resources myResources = getResources();
```

Класс Resources содержит геттеры для всех доступных видов ресурсов. Принцип его работы заключается в передаче идентификатора того ресурса, чей экземпляр вы хотите получить. В следующем фрагменте кода показан пример использования вспомогательных методов для получения выборки значений из ресурсов:

```
Resources myResources = getResources();

CharSequence styledText = myResources.getText(R.string.stop_message);
```

```
Drawable icon = myResources.getDrawable(R.drawable.app_icon);

int opaqueBlue = myResources.getColor(R.color.opaque_blue);

float borderWidth = myResources.getDimension(R.dimen.standard_border);

Animation tranOut;
tranOut = AnimationUtils.loadAnimation(this, R.anim.spin_shrink_fade);

String[] stringArray;
stringArray = myResources.getStringArray(R.array.string_array);

int[] intArray = myResources.getIntArray(R.array.integer_array);
```

Ресурс, содержащий пошаговую анимацию, возвращается в виде объекта AnimationResources. Вы также можете вернуть значение с помощью метода getDrawable и привести его к соответствующему типу:

```
AnimationDrawable rocket;
rocket = (AnimationDrawable)myResources.getDrawable(R.drawable.frame_by_
frame);
```

Вложенные ресурсы

Можно использовать ссылки на ресурсы в качестве значений для атрибутов внутри других ресурсов, особенно в случае с разметкой и стилями, поскольку это позволяет создавать специальные варианты визуальных тем, локализованных строк и графических объектов. Такой подход также помогает при оптимизации различных изображений и разделителей внутри разметки для экранов с разными размерами и разрешениями.

Чтобы сослаться на один ресурс внутри другого, используйте символ @ и запись следующего вида:

```
attribute="@[packagename:]resourcetype/resourceidentifier"
```

ПРИМЕЧАНИЕ

Android предполагает, что вы используете ресурсы из пакета со своим приложением, поэтому полное имя необходимо указывать только для сторонних пакетов.

В листинге 3.4 представлена разметка, в которой используются цвет, размеры и строки.

Листинг 3.4. Использование ресурсов внутри разметки

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
```

Продолжение ☞

Листинг 3.4 (продолжение)

```
    android:layout_height="fill_parent"
    android:padding="@dimen/standard_border">
    <EditText
      android:id="@+id/myEditText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/stop_message"
      android:textColor="@color/opaque_blue"
    />
</LinearLayout>
```

Использование системных ресурсов

Стандартные приложения в Android имеют множество внешних ресурсов и предоставляют доступ к различным изображениям, экземплярам анимации и разметки, стилям и строкам. Все это вы можете использовать в своих приложениях.

Получение доступа к системным ресурсам внутри кода программы ничем не отличается от того, который был описан в предыдущем разделе. Разница лишь в том, что вместо прикладного класса R вам необходимо задействовать системный класс android.R. В следующем фрагменте кода с помощью метода getString, принадлежащего контексту приложения, извлекается сообщение об ошибке, хранящееся в системных ресурсах:

```
CharSequence httpError = getString(android.R.string.httpErrorBadUrl);
```

Чтобы получить доступ к системным ресурсам внутри XML-файла, укажите в качестве пакета значение android:

```
<EditText
  android:id="@+id/myEditText"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@android:string/httpErrorBadUrl"
  android:textColor="@android:color/darker_gray"
/>
```

Размещение ссылок на визуальные стили в текущей теме

Использование визуальных тем — отличный способ обеспечить целостность пользовательского интерфейса вашего приложения. Вместо того чтобы полностью описывать каждый стиль, Android предоставляет ссылки, с помощью которых вы можете использовать стили из текущей темы.

Чтобы задать ссылку на ресурс, который нужно применить, вместо символа @ укажите префикс ?android:. Ниже приведен фрагмент кода из предыдущего примера, в котором цвет текста берется из текущей темы, а не из внешнего ресурса:

```
<EditText
  android:id="@+id/myEditText"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/stop_message"
  android:textColor="?android:textColor"
/>
```

Такой подход позволяет создавать визуальные стили, которые смогут меняться по мере преобразований в текущей теме, и избавляет от необходимости редактировать каждый отдельный ресурс.

Ресурсы для приложения To-Do List

В этом примере вы создадите новые внешние ресурсы, подготовите почву для добавления функциональности в приложение To-Do List, работу над которым начали в главе 2. Созданные строковые и растровые ресурсы будут использованы в главе 4 при реализации системы меню.

Далее вы познакомитесь с ходом создания ресурсов, содержащих текст и значки для пунктов меню Add и Remove, а также научитесь описывать визуальную тему и использовать ее в приложении.

1. Создайте два новых изображения в формате PNG: одно будет символизировать добавление элементов в список задач, другое — их удаление. Каждое изображение должно иметь размеры около 16 × 16 пикселов (как показано на рис. 3.5).

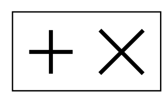


Рис. 3.5.

2. Скопируйте изображения в каталог res/drawable-mdpi и обновите проект.

3. Откройте ресурс `strings.xml` из каталога res/values и добавьте значения для пунктов меню add_new, remove и cancel (заодно можете удалить стандартное строковое значение с приветствием).

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">To Do List</string>
  <string name="add_new">Add New Item</string>
  <string name="remove">Remove Item</string>
  <string name="cancel">Cancel</string>
</resources>
```

4. Создайте новую тему для приложения, добавив в каталог res/values новый файл с ресурсом под названием `styles.xml`. Тема должна

основываться на стандартной, но при этом устанавливать свой размер шрифта для стандартного текста.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="ToDoTheme"parent="@android:style/Theme.Black">
    <item name="android:textSize">12sp</item>
  </style>
</resources>
```

5. Примените тему к своему проекту в манифесте.

```
<activity android:name=".ToDoList"
          android:label="@string/app_name"
          android:theme="@style/ToDoTheme">
```

Создание ресурсов для разных языковых настроек и аппаратных конфигураций

Одна из наиболее очевидных выгод от применения внешних ресурсов — возможность использовать механизм динамического выбора ресурсов, доступный в Android.

Пользуясь структурой каталогов, описанной далее, вы можете создавать ресурсы со значениями для конкретных языков, регионов и аппаратных конфигураций. Во время выполнения программы Android сам динамически выберет подходящие значения.

Вы можете указывать альтернативные значения, применяя параллельные структуры каталогов внутри директории res. Дефис (-) используется для выделения спецификаторов, с помощью которых задаются условия для альтернативных ресурсов.

В следующем примере показано дерево каталогов, которое наряду со стандартными строковыми значениями хранит варианты для французского языка и франкоканадского региона:

```
Project/
  res/
    values/
      strings.xml
    values-fr/
      strings.xml
    values-fr-rCA/
      strings.xml
```

Перечислим спецификаторы, которые можно применить для предоставления альтернативных значений в ресурсах.

- **Мобильный код страны и код мобильного оператора (MCC/MNC).** Информация о стране и опционально о мобильной сети привязана

к SIM-карте устройства. MCC состоит из символов mcc, за которыми следует трехзначный код страны. При желании можете добавить MNC, используя символы mnc и двузначный код мобильной сети (например, mcc234-mnc20 или mcc310). Список кодов MCC/MNC предлагает Википедия по адресу http://en.wikipedia.org/wiki/Mobile_Network_Code.

- **Язык и регион.** Язык указывается с помощью языкового кода в формате ISO 639-1, состоящего из двух символов в нижнем регистре. В случае необходимости за ним может следовать обозначение региона в виде буквы r и двухсимвольного кода в формате ISO 3166-1-alpha-2, записанного в верхнем регистре (например, en, en-rUS или en-rGB).
- **Размер экрана.** Может иметь одно из следующих значений: small (меньше, чем HVGA), medium (HVGA или меньше, чем VGA) или large (VGA или больше).
- **Высота и ширина экрана.** Указывайте значения long или notlong, если хотите создать ресурсы, предназначенные специально для широкоформатных дисплеев (например, long для WVGA, notlong для QVGA).
- **Ориентация экрана в пространстве.** Поддерживаются режимы port (портретный), land (альбомный) и square (если высота экрана равна ширине).
- **Плотность пикселов на экране.** Плотность пикселов, измеряемая в точках на дюйм (dpi). Для экранов с низкой (120 dpi), средней (160 dpi) и высокой (240 dpi) плотностью рекомендуется использовать значения ldpi, mdpi и hdpi соответственно. Вы можете указывать значение nodpi для растровых ресурсов, которые не должны масштабироваться, чтобы поддерживать конкретную плотность пикселов. При выборе именно этого типа ресурсов (в отличие от остальных) система не требует точного совпадения. Подбирая подходящий каталог, Android остановится на спецификаторе, который точнее всего описывает плотность пикселов экрана устройства, и откорректирует масштаб результирующего объекта Drawable соответствующим образом.
- **Тип сенсорного экрана:** notouch, stylus или finger.
- **Наличие клавиатуры:** keysexposed, keyshidden или keyssoft.
- **Тип ввода**, поддерживаемый клавиатурой: nokeys, qwerty или 12key.
- **Способ навигации** по пользовательскому интерфейсу: nonav, dpad, trackball или wheel.

Вы можете задать несколько спецификаторов для любого типа ресурсов, разделив их дефисами. Поддерживаются любые сочетания, однако они должны использоваться в порядке из вышеприведенного списка. В одном спецификаторе может применяться не более одного значения.

Далее показаны корректные и некорректные имена каталогов для альтернативных ресурсов Drawable:

- корректные:
  - `drawable-en-rUS`;
  - `drawable-en-keyshidden`;
  - `drawable-long-land-notouch-nokeys`;
- некорректные:
  - `drawable-rUS-en` (не в том порядке);
  - `drawable-rUS-rUK` (несколько значений для одного спецификатора).

При извлечении ресурсов во время выполнения программы Android найдет самый подходящий вариант из всех доступных. Пройдясь по списку каталогов, где хранятся нужные значения, он выберет тот, у которого больше всего совпавших спецификаторов. Если требованиям соответствуют сразу два каталога, учитывается порядок совпавших спецификаторов из предыдущего списка.

ПРИМЕЧАНИЕ

Если на заданном устройстве не обнаружено соответствующих ресурсов, при попытке получения доступа к ним ваша программа сгенерирует исключение. Чтобы избежать подобной ситуации, необходимо всегда добавлять значения по умолчанию для всех типов ресурсов в каталог без спецификаторов.

Изменение конфигурации во время выполнения программы

При изменении языка, региона или аппаратной конфигурации Android прерывает работу всех приложений и затем запускает их повторно, перезагружая значения из ресурсов.

Подобное поведение не всегда уместно и желательно. Например, некоторые изменения конфигурации (ориентация экрана в пространстве, доступность клавиатуры) могут произойти только лишь из-за того, что пользователь повернул устройство или выдвинул клавиатуру. Вы можете настраивать, каким образом ваше приложение будет реагировать на подобные изменения, обнаруживая их и выполняя собственные действия.

Чтобы заставить Активность отслеживать изменения конфигурации при выполнении программы, добавьте в ее узел в манифесте атрибут `android:configChanges`, указав, какие именно события хотите обрабатывать.

Перечислим значения, с помощью которых можно описать изменения конфигурации:

- orientation — положение экрана изменено с портретного на альбомное (или наоборот);
- keyboardHidden — клавиатура выдвинута или спрятана;
- fontScale — пользователь изменил предпочтительный размер шрифта;
- locale — пользователь выбрал новые языковые настройки;
- keyboard — изменился тип клавиатуры; например, телефон может иметь 12-клавишную панель, при повороте которой появляется полноценная клавиатура;
- touchscreen или navigation — изменился тип клавиатуры или способ навигации. Как правило, такие события не встречаются.

В некоторых случаях одновременно будут срабатывать несколько событий. Например, когда пользователь выдвигает клавиатуру, большинство устройств генерируют события keyboardHidden и orientation.

Вы можете выбирать несколько событий, которые хотите обрабатывать самостоятельно, разделяя их символом |.

В листинге 3.5 показан узел Активности, в котором декларируется обработка событий, связанных с ориентацией экрана в пространстве и доступностью клавиатуры.

Листинг 3.5. Описание Активности для обработки динамических изменений ресурсов

```
<activity android:name=".TodoList"
          android:label="@string/app_name"
          android:theme="@style/TodoTheme"
          android:configChanges="orientation|keyboardHidden"/>
```

Наличие атрибута `android:configChanges` отменяет перезапуск приложения при заданных изменениях конфигурации. Вместо этого внутри Активности срабатывает метод onConfigurationChanged. Переопределите его, чтобы появилась возможность обрабатывать изменения в конфигурации. Используйте переданный объект Configuration, чтобы получить новые значения, как показано в листинге 3.6. Не забудьте вызвать одноименный метод из родительского класса и перезагрузить измененные значения со всех ресурсов, которые используются внутри Активности.

Листинг 3.6. Обработка изменения конфигурации внутри кода программы

```
@Override
public void onConfigurationChanged(Configuration _newConfig) {

  super.onConfigurationChanged(_newConfig);

  [ ... Обновите пользовательский интерфейс, используя данные
```

Продолжение ⇗

Листинг 3.6 (продолжение)

```
из ресурсов ... ]

  if (_newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
    [ ... Реакция на измененную ориентацию экрана ... ]
  }

  if (_newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {
    [ ... Реакция на выдвигание/задвигание клавиатуры ... ]
  }
}
```

На момент вызова обработчика onConfigurationChanged данные из ресурсов, используемых внутри Активности, уже обновлены и имеют актуальные значения, поэтому применять их безопасно.

Любые изменения конфигурации, которые не были явно помечены для обработки внутри вашего приложения, приведут к перезапуску Активности, минуя вызов метода onConfigurationChanged.

Знакомство с классом Application в Android

Создание собственной реализации класса Application дает возможность:

- контролировать состояние приложения;
- передавать объекты между программными компонентами;
- поддерживать и контролировать ресурсы, которые используются в нескольких компонентах одного приложения.

Вместе с процессом программы создается экземпляр класса Application, который вы описали и зарегистрировали в манифесте. Исходя из этого ваш класс Application по природе синглтон (singleton) и должен регистрироваться соответствующим образом, предоставляя доступ к методам и свойствам.

Наследование и использование класса Application

В листинге 3.7 показан каркас для наследования класса Application и реализации его в качестве синглтона.

Листинг 3.7. Каркас для класса Application

```
import android.app.Application;
import android.content.res.Configuration;

public class MyApplication extends Application {

  private static MyApplication singleton;

  // Возвращает экземпляр данного класса
```

```
    public static MyApplication getInstance() {
      return singleton;
    }

    @Override
    public final void onCreate() {
      super.onCreate();
      singleton = this;
    }
}
```

Создав новый класс Application, нужно зарегистрировать его внутри тега `<application>` в манифесте:

```
<application android:icon="@drawable/icon"
             android:name="MyApplication">
  [... Вложенные в манифест теги ...]
</application>
```

При запуске приложения создается экземпляр вашей реализации класса Application. Добавляйте новые свойства для хранения программного состояния и глобальных ресурсов, чтобы иметь возможность получить к ним доступ из компонентов приложения:

```
MyObject value = MyApplication.getInstance().getGlobalStateValue();
MyApplication.getInstance().setGlobalStateValue(myObjectValue);
```

Такой подход особенно эффективен при передаче объектов между слабосвязанными частями вашей программы, а также для контроля за общими ресурсами и состоянием приложения.

Переопределение событий, связанных с жизненным циклом приложения

Класс Application — обработчики событий, отвечающие за создание и завершение работы приложения, поведение программы при нехватке памяти и изменениях конфигурации (описаны в предыдущем разделе).

Переопределяя эти методы, вы можете предусмотреть поведение приложения в подобных ситуациях.

- onCreate. Вызывается при создании приложения, переопределяется для инициализации синглтона программы и создания и инициализации свойств, в которых хранятся состояния приложения или общие ресурсы.
- onTerminate. Может быть вызван при преждевременном завершении работы объекта Application. Хотя нет никакой гарантии, что так все и будет, имейте это в виду. Если работа приложения прерывается ядром, чтобы освободить ресурсы для других программ, процесс

завершится без предупреждения и без вызова метода onTerminate из объекта Application.

- onLowMemory. Предоставляет возможность хорошо оптимизированным приложениям освобождать дополнительную память, когда система работает в условиях ограниченных ресурсов. Как правило, вызывается в ситуациях, когда фоновые процессы уже закрыты, а памяти для приложений, находящихся на переднем плане, все еще не хватает. Переопределяйте этот метод, чтобы очищать кэш или освобождать ненужные ресурсы.
- onConfigurationChanged. В отличие от объектов Activity ваше приложение не перезапускается при изменениях конфигурации. Переопределяйте этот метод, если вам необходимо обрабатывать изменения конфигурации на уровне приложения.

Как видно из листинга 3.8, в процессе переопределения этих обработчиков всегда нужно вызывать одноименные методы родительского класса.

Листинг 3.8. Переопределение обработчиков, связанных с жизненным циклом приложения

```
public class MyApplication extends Application {

  private static MyApplication singleton;

  // Возвращает экземпляр данного класса
  public static MyApplication getInstance() {
    return singleton;
  }

  @Override
  public final void onCreate() {
    super.onCreate();
    singleton = this;
  }

  @Override
  public final void onTerminate() {
    super.onTerminate();
  }

  @Override
  public final void onLowMemory() {
    super.onLowMemory();
  }

  @Override
  public final void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
  }
}
```

Детальный обзор Активностей в Android

При создании экранов с графическим интерфейсом наследуется класс Activity и используются Представления для взаимодействия с пользователем.

Каждая Активность — экран (по аналогии с Формой), который приложение может показывать пользователям. Чем сложнее приложение, тем больше экранов нужно.

Создавайте Активности для каждого экрана, который хотите отобразить. Речь как минимум о начальном экране, который отвечает за работу основного пользовательского интерфейса приложения. Такой интерфейс часто дополняется второстепенными Активностями, предназначенными для ввода информации, ее вывода, предоставления дополнительных возможностей. Запуск новой Активности (или возврат из нее) инициирует перемещение между экранами.

Большинство Активностей проектируются таким образом, чтобы занимать весь экран, но вы можете создавать полупрозрачные или плавающие диалоговые окна.

Создание Активности

Наследуйте класс Activity, чтобы создать новую Активность. Внутри этого нового класса необходимо определить пользовательский интерфейс и реализовать нужную функциональность. Базовый каркас для новой Активности показан в листинге 3.9.

Листинг 3.9. Каркас для Активности

```
package com.paad.myapplication;

import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity {

  /** Вызывается при первом создании Активности. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

Базовый класс Activity представляет собой пустой экран, который инкапсулирует всю работу по отображению окна. Пустая Активность не особо нужна, поэтому первое, что необходимо сделать, — создать пользовательский интерфейс с помощью Представлений и разметки.

Представления — элементы управления, которые отображают данные и обеспечивают взаимодействие с пользователем. Android предоставляет несколько классов разметки, наследников ViewGroup, которые могут содержать Представления для проектирования пользовательских интерфейсов.

Подробно Представления и объекты ViewGroup рассматриваются в главе 4, из нее вы также узнаете, какие элементы и как использовать, а также как создавать собственные Представления и экземпляры разметки.

Чтобы назначить пользовательский интерфейс для Активности, внутри обработчика onCreate вызовите метод setContentView, принадлежащий объекту Activity.

В этом первом фрагменте в качестве пользовательского интерфейса для Активности выступает элемент TextView:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  TextView textView = new TextView(this);
  setContentView(textView);
}
```

В будущем, скорее всего, вам потребуются более сложные интерфейсы. Вы можете создавать разметку как в коде программы, используя объекты ViewGroup, так и с помощью внешних ресурсов, передавая соответствующий идентификатор. Этот стандартный для Android подход показан в следующем фрагменте:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
}
```

Чтобы использовать Активность в своем приложении, зарегистрируйте ее в манифесте. Добавьте новый тег <activity> внутри узла <application>. Данный тег включает атрибуты для хранения метаинформации: метку, значок, необходимые полномочия и визуальные темы, используемые Активностью. Без соответствующего тега <activity> Активность не может быть выведена на экран.

Ниже на примере XML-кода показано, как добавить тег для класса MyActivity, созданного в листинге 3.9.

Листинг 3.10. Описание Активности в формате XML

```
<activity android:label="@string/app_name"
          android:name=".MyActivity">
</activity>
```

Внутри тега <activity> вы можете добавлять узлы <intent-filter>, описывающие Намерения, которые ваша Активность станет отслеживать для дальнейшего взаимодействия. Каждый Фильтр намерений определяет одно или несколько действий и категорий, которые поддерживаются Активностью. Намерения и их Фильтры подробно рассматриваются в главе 5, но стоит отметить, что Активность может быть доступна из главного меню запуска приложений только в том случае, если она содержит тег <intent-filter> для действия MAIN и имеет категорию LAUNCHER. На это обращено внимание в листинге 3.11.

Листинг 3.11. Описание главной Активности для приложения

```
<activity android:label="@string/app_name"
          android:name=".MyActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Жизненный цикл Активности

Четкое понимание жизненного цикла Активности важно для создания приложений, которые должны правильно управлять ресурсами и оставлять целостное впечатление.

Как уже говорилось ранее, приложения в Android не контролируют жизненный цикл своих процессов, система сама управляет всеми программами и, как следствие, Активностями внутри них.

Во время контролирования процесса (в том числе при завершении его работы) состояние принадлежащей ему Активности помогает системе определить приоритет родительского приложения. Приоритет приложения, в свою очередь, влияет на то, с какой вероятностью его работа (а также работа Активностей внутри него) будет прервана.

Стеки (очереди) Активностей

Состояние каждой Активности определяется ее позицией в соответствующей очереди — стеке (last-in-first-out) объектов Activity, запущенных в данный момент. При запуске новой Активности экран, находившийся на переднем плане, перемещается на вершину стека. Если пользователь вернется обратно с помощью кнопки Назад или работающий на переднем плане объект Activity закроется, следующая Активность в стеке перемещается вверх и становится текущей. Этот процесс показан на рис. 3.6.

Как уже говорилось, на приоритет приложения влияет его самая приоритетная Активность. Когда диспетчер памяти в Android решает, какую

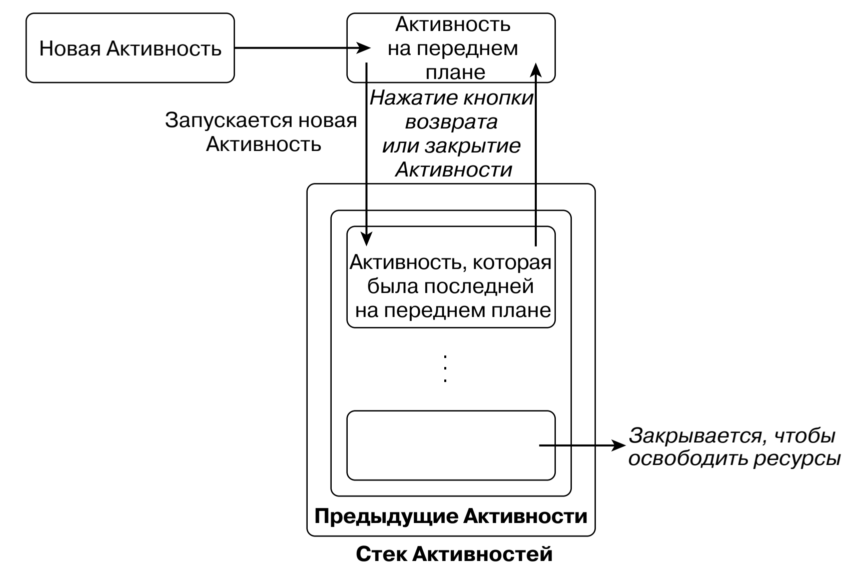


Рис. 3.6.

программу закрыть, чтобы освободить ресурсы, он использует этот стек для определения приоритетов приложений на основе их Активностей.

Состояния Активностей

Создавая объекты Activity, система помещает их в стек, (см. рис. 3.6). При уничтожении эти объекты оттуда убираются, проходя через четыре возможных состояния.

- **Активное.** Когда Активность на вершине стека, она видна и выходит на передний план, имея возможность принимать пользовательский ввод. Android будет пытаться любой ценой сохранить ее в рабочем состоянии, при необходимости прерывая работу других Активностей, которые находятся на более низких позициях в стеке, обеспечивая для нее все необходимые ресурсы. Ее работа будет приостановлена, если на передний план выйдет другая Активность.
- **Приостановленное.** В некоторых ситуациях ваша Активность будет видна на экране, но не сможет принимать пользовательский ввод: в этот момент она приостановлена. Такое состояние наступает, когда полупрозрачные или плавающие диалоговые окна становятся активными и частично ее перекрывают. В приостановленном виде Активность рассматривается как полноценно работающая, однако она не может взаимодействовать с пользователем. Ее работа может преждевременно завершиться, если системе нужно выделить ресурсы для той Актив-

ности, что на переднем плане. При полном исчезновении с экрана Активность останавливается.

- **Остановленное.** Когда Активность перестает быть видимой, она останавливается и остается в памяти, сохраняя всю информацию о своем состоянии. То есть становится кандидатом на преждевременное закрытие, если системе понадобится память для чего-нибудь другого. При остановке Активности важно сохранить данные и текущее состояние пользовательского интерфейса. Как только объект Activity завершает свою работу или закрывается, он становится неактивным.
- **Неактивное.** Это состояние наступает после того, как работа объекта Activity была завершена, и перед тем, как он будет запущен снова. Такая Активность удаляется из стека и должна запускаться повторно, чтобы ее можно было вывести на экран и снова использовать.

Смена состояний — недетерминированный процесс и полностью управляется системным диспетчером памяти. Сперва Android закроет приложения, содержащие объекты Activity в неактивном состоянии, потом перейдет к тем, чьи Активности остановлены или приостановлены (только в крайних случаях).

ПРИМЕЧАНИЕ

Чтобы приложение вызывало целостное впечатление, пользователь не должен видеть процесс смены его состояний. Меняя свое состояние с приостановленного на остановленное или с неактивного на активное, объект Activity внешне не должен меняться. Поэтому при остановке или приостановлении работы Активности важно сохранять состояние пользовательского интерфейса и все данные. Как только Активность выходит на передний план, она должна восстановить все сохраненные значения.

Отслеживание изменений текущего состояния

Чтобы экземпляры Активностей могли реагировать на изменения текущего состояния, Android предоставляет набор обработчиков для событий, возникающих при переходе объекта Activity через стадии полноценный, видимый и активный. Они с учетом состояний, описанных в предыдущем разделе, показаны на рис. 3.7.

В листинге 3.12 приведен каркас с заглушками для методов внутри Активности, которые обрабатывают изменения состояний. Комментарии к каждой такой заглушке описывают действия, которые нужно учитывать при обработке этих событий.

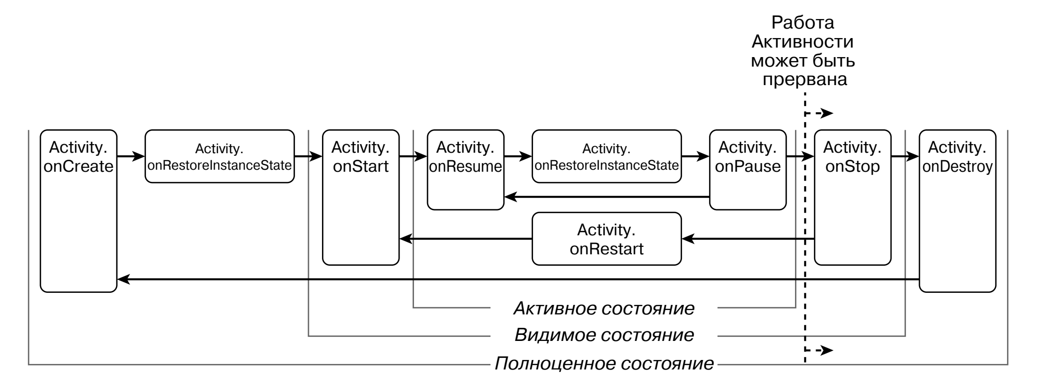


Рис. 3.7.

Листинг 3.12. Обработчики, следящие за сменой состояний Активности

```
package com.paad.myapplication;

import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity {

  // Вызывается при входе в "полноценное" состояние.
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Инициализируйте Активность.
  }

  // Вызывается, когда метод onCreate завершил свою работу,
  // и используется для восстановления состояния пользовательского
  // интерфейса
  @Override
  public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    // Восстановите состояние UI из переменной savedInstanceState.
    // Этот объект типа Bundle также был передан в метод onCreate.
  }

  // Вызывается перед тем, как Активность становится "видимой".
  @Override
  public void onRestart(){
    super.onRestart();
    // Загрузите изменения, учитывая то, что Активность
    // уже стала "видимой" в рамках данного процесса.
  }

  // Вызывается в начале "видимого" состояния.
  @Override
  public void onStart(){
    super.onStart();
```

```
    // Примените к UI все необходимые изменения, так как
    // Активность теперь видна на экране.
  }

  // Вызывается в начале "активного" состояния.
  @Override
  public void onResume(){
    super.onResume();
    // Возобновите все приостановленные обновления UI,
    // потоки или процессы, которые были "заморожены",
    // когда данный объект был неактивным.
  }

  // Вызывается для того, чтобы сохранить пользовательский интерфейс
  // перед выходом из "активного" состояния.
  @Override
  public void onSaveInstanceState(Bundle savedInstanceState) {
    // Сохраните состояние UI в переменную savedInstanceState.
    // Она будет передана в метод onCreate при закрытии и
    // повторном запуске процесса.
    super.onSaveInstanceState(savedInstanceState);
  }

  // Вызывается перед выходом из "активного" состояния
  @Override
  public void onPause(){
    // "Замораживает" пользовательский интерфейс, потоки
    // или трудоемкие процессы, которые могут не обновляться,
    // пока Активность не находится на переднем плане.
    super.onPause();
  }

  // Вызывается перед тем, как Активность перестает быть "видимой".
  @Override
  public void onStop(){
    // "Замораживает" пользовательский интерфейс, потоки
    // или операции, которые могут подождать, пока Активность
    // не отображается на экране. Сохраняйте все введенные
    // данные и изменения в UI так, как будто после вызова
    // этого метода процесс должен быть закрыт.
    super.onStop();
  }

  // Вызывается перед выходом из "полноценного" состояния.
  @Override
  public void onDestroy(){
    // Очистите все ресурсы. Это касается завершения работы
    // потоков, закрытия соединений с базой данных и т. д.
    super.onDestroy();
  }
}
```

Как видно из кода, переопределяя эти обработчики, вы всегда должны вызывать одноименные методы родительского класса.

Подробнее о состояниях Активности

Пребывая в «полноценном» состоянии, между созданием и уничтожением, объект Activity может несколько раз становится активным и видимым. Каждый такой переход сопровождается вызовом ранее описанных обработчиков. В этом разделе более подробно рассматривается каждое из состояний, а также характерные для них события.

Полноценное состояние. Активность пребывает в полноценном состоянии между первым вызовом метода onCreate и последним вызовом метода onDestroy. В некоторых ситуациях работа Активности может быть прервана без вызова onDestroy.

Используйте обработчик onCreate, чтобы инициализировать Активность, загружайте пользовательский интерфейс, размещайте ссылки на свойства класса, связывайте данные с элементами управления, создавайте Сервисы и потоки. Метод onCreate принимает объект Bundle, содержащий состояние пользовательского интерфейса, сохраненное в последнем вызове обработчика onSaveInstanceState. Для восстановления графического интерфейса в его предыдущем состоянии нужно задействовать эту переменную: внутри onCreate или переопределяя метод onRestoreInstanceState.

Обратитесь к переопределенному обработчику onDestroy, чтобы очищать любые ресурсы, созданные в методе onCreate, и закрывать подключения к таким внешним источникам, как сеть или базы данных.

В официальном руководстве по написанию эффективного кода для Android рекомендуется избегать создания объектов с коротким жизненным циклом. Быстрое появление и уничтожение объектов провоцирует дополнительное срабатывание сборщика мусора (garbage collector, или GC), что может повлиять на впечатления пользователя от работы приложения. Если ваша Активность регулярно выделяет память для одного и того же набора объектов, подумайте, можно ли переместить их создание в метод onCreate, который вызывается всего один раз за все время жизни объекта Activity.

Видимое состояние. Активность считается видимой между вызовами методов onStart и onStop. В таком состоянии она видна пользователю, но при этом может быть и частично перекрытой и не реагировать на ввод. Переходя из фонового режима на передний план и обратно, Активности могут несколько раз менять свое «видное» состояние, находясь при этом в полноценной фазе жизненного цикла. Хотя обычно это не происходит, но в крайних случаях Android может прервать работу видимой Активности без вызова метода onStop.

Обработчик onStop используется для приостановки анимации, потоков, отслеживания показаний датчиков, запросов к GPS, таймеров, Сервисов или других процессов, которые нужны исключительно для обновления

пользовательского интерфейса. Нет смысла потреблять ресурсы (такты центрального процессора или сетевой трафик) для обновления интерфейса, в то время как он не виден на экране. Примените методы onStart или onRestart для возобновления или повторного запуска этих процессов, когда Активность опять станет видимой.

onRestart предшествует вызовам метода onStart (кроме самого первого). Используйте его для специальных действий, которые должны выполняться только при повторном запуске Активности в рамках «полноценного» состояния.

Методы onStart/onStop также нужны для регистрации (и ее отмены) Широковещательных приемников, которые задействуются исключительно для обновления графического интерфейса. Больше о Широковещательных приемниках вы узнаете в главе 5.

Активное состояние. Оно начинается с момента вызова метода onResume и заканчивается при вызове onPause. В этом состоянии Активность находится на переднем плане и может принимать пользовательский ввод. Прежде чем быть уничтоженным, объект Activity способен несколько раз входить в активное состояние и выходить из него, например при потере фокуса, отображении новой Активности или когда устройство входит в режим ожидания. Пытайтесь размещать в методах onPause и onResume относительно быстрый и легковесный код, чтобы ваше приложение оставалось отзывчивым при скрытии с экрана или выходе на передний план.

Прямо перед onPause вызывается обработчик onSaveInstanceState. Он предоставляет возможность сохранять состояние пользовательского интерфейса Активности в объект Bundle, который потом будет передаваться в методы onCreate и onRestoreInstanceState. Используйте обработчик onSaveInstanceState для сохранения состояния интерфейса (например, состояния флажков, текущего выделенного элемента или введенных, но не сохраненных данных), чтобы объект Activity при следующем входе в активное состояние мог вывести на экран тот же UI. Рассчитывайте, что перед завершением работы процесса во время активного состояния будут вызваны обработчики onSaveInstanceState и onPause.

В большинстве реализаций класса Activity переопределяется как минимум метод onPause. Внутри него фиксируются несохраненные изменения, потому как после его выполнения работа Активности может прерваться без предупреждения. Исходя из архитектуры своего приложения, вы также можете приостановить выполнение потоков, процессов или Широковещательных приемников, пока Активность не появится на переднем плане.

Метод onResume может быть довольно легковесным. Вам не нужно перезагружать состояние пользовательского интерфейса внутри него, так как эти функции возложены на обработчики onCreate и onRestoreInstanceState.

Используйте метод onResume для регистрации любых Широковещательных приемников или других процессов, которые должны приостанавливаться внутри обработчика onPause.

Классы Activity в Android

Android SDK включает набор классов, наследованных от Activity. Они предназначены для упрощения работы с виджетами, которые часто встречаются в обычном пользовательском интерфейсе. Перечислим некоторые из них (наиболее полезные).

- MapActivity. Инкапсулирует обработку ресурсов, необходимых для поддержки элемента MapView внутри Активности. Больше информации о классах MapActivity и MapView в главе 8.
- ListActivity. Обертка для класса Activity, главная особенность которой — виджет ListView, привязанный к источнику данных, и обработчики, срабатывающие при выборе элемента из списка.
- ExpandableListActivity. То же самое, что и ListActivity, но вместо ListView поддерживает ExpandableListView.
- TabActivity. Позволяет разместить несколько Активностей или Представлений в рамках одного экрана, используя вкладки для переключения между элементами.

Резюме

В этой главе вы узнали, как разрабатывать надежные приложения, используя слабосвязанные компоненты: Активности, Сервисы, Источники данных, Намерения и Широковещательные приемники, объединяемые программным манифестом.

Вы получили представление о жизненном цикле приложений в Android, узнали, каким образом их приоритет зависит от состояния процесса (которое, в свою очередь, определяется состоянием его компонентов).

Чтобы в полной мере воспользоваться преимуществами доступных устройств и распространенностью Android во многих странах мира, научились создавать внешние ресурсы и описывать альтернативные значения для конкретных регионов, языков и аппаратных конфигураций.

Вы также познакомились с классом Application, узнали, как его расширять, сделав тем самым управление состоянием приложения проще, равно как и передачу данных между его составными частями.

После вы более подробно рассмотрели объекты Activity и их роль в иерархии программных компонентов, узнали, как создавать новые Активности и изучили их жизненный цикл. В частности, речь шла о переходах между

разными состояниями и о том, как их отслеживать, чтобы впечатление от использования приложения было целостным.

Под конец главы вы познакомились с некоторыми специализированными классами, наследованными от Activity.

В следующей главе вы научитесь создавать пользовательские интерфейсы с помощью разметки и некоторых стандартных виджетов. Узнаете, как расширять, изменять и группировать Представления, создавая специализированные виджеты. Сможете разработать свои собственные уникальные элементы интерфейса с чистого листа, познакомитесь с системой меню, которую предоставляет Android.

Глава 4
СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Содержание главы

- Использование Представлений и менеджеров компоновки.
- Оптимизация разметки.
- Ресурсы Drawable, хранящиеся в формате XML.
- Создание пользовательских интерфейсов, не зависящих от разрешения экрана.
- Система меню в Android.
- Создание и использование Представлений, их расширение и группировка.

Очень важно, чтобы вы создавали для своих приложений симпатичные и интуитивно понятные пользовательские интерфейсы. Главным приоритетом при проектировании UI помимо функциональности должно быть обеспечение изящности и простоты в использовании.

Стивен Фрай о значении стиля как одной из важнейших составляющих дизайна цифровых устройств говорит следующее:

Как будто устройство, напрочь лишенное изящества, может работать! Как будто устройство, в работе которого есть недостатки, можно назвать изящным... Да, внешний вид имеет значение. Еще как имеет. Это не просто что-то наносное или необязательное, это и есть сам продукт (Стивен Фрай, The Guardian (27.10.2007)).

С увеличением размеров и разрешений экранов, с ростом мощности процессоров мобильные приложения становятся все более визуально насыщенными. Хотя миниатюрные дисплеи вызывают множество трудностей при проектировании сложных визуальных интерфейсов, учитывая широкую распространенность мобильных устройств, оно того стоит.

В этой главе вы изучите основные графические элементы, предоставляемые системой Android, и узнаете, как использовать Представления, Группы представлений (View Groups) и разметку, чтобы создавать функциональные

и интуитивно понятные пользовательские интерфейсы для своих Активностей.

Познакомившись с некоторыми элементами управления, доступными в Android SDK, вы научитесь их расширять и настраивать. Узнаете, как с помощью объектов ViewGroup объединять Представления для создания легковесных, рассчитанных на многократное применение элементов пользовательского интерфейса, состоящих из отдельных частей и взаимодействующих между собой. Вы также научитесь создавать свои собственные Представления, воплощая в жизнь новые подходы к отображению данных и к взаимодействию с пользователями.

Отдельные элементы пользовательского интерфейса в Android располагаются на экране с помощью целого ряда объектов, наследованных от ViewGroup. Правильное использование разметки чрезвычайно важно при создании хороших интерфейсов. В этой главе вы познакомитесь с несколькими стандартными классами разметки и научитесь ими пользоваться, также узнаете, как создать собственный менеджер компоновки.

С ростом количества новых устройств на базе Android увеличивается и разнообразие размеров и разрешений экранов, на которых приложение может быть запущено. Вы узнаете, как создавать разметку и объекты Drawable, не зависящие от разрешения экрана. Благодаря рекомендациям по разработке и тестированию ваши пользовательские интерфейсы будут отлично выглядеть на любом дисплее.

Механизм управления главным и контекстным меню в приложениях для Android обеспечивает новый подход, оптимизированный для современных сенсорных устройств. При знакомстве с моделью пользовательского интерфейса в Android окончание этой главы посвящено созданию и использованию главных и контекстных меню.

Основы проектирования пользовательского интерфейса в Android

Пользовательский интерфейс (user interface, UI), впечатления от использования (user experience, UX), взаимодействие человека с компьютером (human computer interaction, HCI), юзабилити (usability) — весьма обширные темы, которые не могут быть полностью раскрыты в этой книге. Тем не менее при создании собственных пользовательских интерфейсов вы должны в них ориентироваться.

С появлением Android введено несколько новых терминов для обозначения уже известных программных абстракций, которые будут подробно рассмотрены в следующих разделах.

- **Представления.** Это базовый класс для всех визуальных элементов интерфейса (более известных как элементы управления или

виджеты). Все элементы UI, включая разметку, — производные от класса View.

- **Группы представлений.** Это потомок класса View, который может содержать внутри себя несколько дочерних Представлений. Наследуйте класс ViewGroup, чтобы создавать сложные Представления, состоящие из взаимосвязанных элементов. Класс ViewGroup также стал основой для менеджеров компоновки, которые помогают размещать элементы управления внутри Активностей.
- **Активности.** Активности, подробно описанные в предыдущей главе, — это отображаемые окна (или экраны). Активность — эквивалент Формы для Android. Чтобы вывести на экран пользовательский интерфейс, необходимо назначить для Активности хотя бы одно Представление (как правило, разметку).

Android предоставляет несколько универсальных элементов пользовательского интерфейса, виджеты и менеджеры компоновки (разметку).

Чтобы создавать собственные уникальные графические приложения, вам, скорее всего, придется расширять и модифицировать стандартные Представления (комбинировать имеющиеся или воплощать совершенно новые).

Знакомство с Представлениями

Как говорилось выше, все визуальные компоненты в Android — потомки класса View и, как правило, называются Представлениями. Вы также будете часто встречать термины «элемент управления» или «виджет» (не путайте с виджетами для домашнего экрана, которые описываются в главе 10) — с ними вы должны быть знакомы, если имеете какой-либо опыт разработки GUI.

ViewGroup — это производный от View класс, спроектированный для хранения нескольких Представлений одновременно. Как правило, Группы представлений используются либо для создания независимых компонентов многократного использования, либо для управления компоновкой дочерних элементов (в таком случае объект ViewGroup, как правило, называется разметкой).

Поскольку все визуальные элементы происходят от класса View, вы, вероятно, столкнетесь с такими его названиями, как «виджет», «элемент управления» и «представление».

Создавая приложение To-Do List в главе 2, вы уже успели познакомиться с разметкой и тремя стандартными Представлениями: LinearLayout, ListView и TextView.

В следующих разделах вы научитесь собирать воедино постоянно усложняющиеся пользовательские интерфейсы. Узнаете, как с помощью наследования Представлений, доступных в SDK, создавать новые структурированные элементы управления, после чего создадите собственное Представление с нуля.

Создание пользовательского интерфейса для Активностей с помощью Представлений

Новая Активность изначально — пустой экран, где вы должны разместить свой пользовательский интерфейс. Для этого необходимо вызвать метод setContentView, передав ему экземпляр Представления или ресурс с разметкой, который вы хотите отобразить. Так как пустой дисплей выглядит не очень вдохновляюще, при переопределении обработчика onCreate почти всегда используется метод setContentView, чтобы назначать пользовательский интерфейс для Активности.

Этот метод принимает либо идентификатор ресурса с разметкой (как было описано в главе 3), либо единственный экземпляр Представления. Благодаря этому вы можете описывать пользовательский интерфейс как внутри кода программы, так и с помощью внешних ресурсов с разметкой, что более предпочтительно.

Используя ресурсы с разметкой, вы отделяете уровень представления от бизнес-логики. Благодаря такому гибкому подходу можно изменять внешний вид, не затрагивая код программы. Это позволяет указывать различные экземпляры разметки, оптимизированные под разные аппаратные конфигурации, вы даже можете менять их во время выполнения программы, основываясь на аппаратных изменениях (например, при повороте экрана).

В листинге 4.1 показано, как с помощью внешнего ресурса с разметкой задать пользовательский интерфейс для Активности. Вы можете получить ссылки на Представления, размещенные внутри разметки, используя метод findViewById. В этом примере предполагается, что файл `main.xml` хранится в каталоге проекта res/layout.

Листинг 4.1. Получение ссылок на элементы разметки внутри Активности

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setContentView(R.layout.main);
  TextView myTextView = (TextView)findViewById(R.id.myTextView);
}
```

Если вы предпочитаете более традиционный подход, то есть возможность создавать пользовательский интерфейс прямо в коде программы.

В листинге 4.2 показано, как назначить новый объект TextView в качестве пользовательского интерфейса.

Листинг 4.2. Создание разметки пользовательского интерфейса в коде программы

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  TextView myTextView = new TextView(this);
  setContentView(myTextView);

  myTextView.setText("Hello, Android");
}
```

Метод setContentView в качестве параметра принимает единственный экземпляр Представления, поэтому чтобы добавить в Активность сразу несколько элементов, необходимо использовать разметку.

Виджеты, доступные в Android

Android — это набор стандартных элементов управления, с помощью которых можно создавать простые интерфейсы. Используя эти Представления (а также изменяя и расширяя их при необходимости), вы можете упростить процесс разработки и обеспечить преемственность между разными приложениями.

Отметим некоторые из наиболее используемых элементов.

- TextView. Стандартная метка, предназначенная для вывода текста. Она поддерживает многострочное отображение, форматирование и автоматический перенос слов и символов.
- EditText. Редактируемое поле для ввода текста. Поддерживает многострочный ввод, перенос слов на новую строку и текст подсказки.
- ListView. Группа представлений, которая формирует вертикальный список элементов, отображая их в виде строк внутри списка. Простейший объект ListView использует TextView для вывода на экран значений toString, принадлежащих элементам массива.
- Spinner. Составной элемент, отображающий TextView в сочетании с соответствующим Представлением ListView, которое позволяет выбрать элемент списка для отображения в текстовой строке. Сама строка состоит из объекта TextView, показывающего текущий выбор, и кнопки, при нажатии которой всплывает диалог выбора.
- Button. Стандартная кнопка, которую можно нажимать.
- CheckBox. Кнопка, поддерживающая два состояния. Представлена в виде отмеченного или неотмеченного флажка.
- RadioButton. Переключатель, поддерживающий два состояния и группировку. Группы таких переключателей пользователь видит как набор

двоичных вариантов, из которых в определенный момент времени может быть выбран только один.

- ViewFlipper. Группа представлений, позволяющая определить набор элементов и горизонтальную строку, в которой одновременно может выводиться только одно Представление. При этом переходы между отображаемыми элементами осуществляются с помощью анимации.
- QuickContactBadge. Выводит на экран эмблему со значком, присвоенным указанному контакту, используя такие данные, как номер телефона, имя, адрес электронной почты или сайта. Нажатие значка приведет к отображению панели, которая предоставляет различные варианты для связи с выбранным контактом, включая звонок, SMS, электронную почту и системы обмена мгновенными сообщениями (IM).

Это только некоторые из доступных виджетов. Android также поддерживает несколько более продвинутых реализаций Представлений, включая элементы для выбора даты и времени, поля ввода с автодополнением, карты, галереи и вкладки. Чтобы получить более полный список доступных виджетов, можете пройти по адресу http://developer.android.com/guide/tutorials/views/index.html.

Рано или поздно вы, как инновационный разработчик, столкнетесь с ситуацией, когда ни один из существующих стандартных элементов управления не подойдет для решения задачи. Ниже в этой главе вы узнаете, как расширять и комбинировать существующие элементы, а также научитесь проектировать и создавать полностью новые виджеты с нуля.

Знакомство с менеджерами компоновки

Менеджер компоновки (более известный как разметка) — это расширение класса ViewGroup, которое используется для позиционирования дочерних элементов внутри пользовательского интерфейса. Экземпляры разметки могут быть вложенными. Комбинируя их, вы можете создавать сколь угодно сложные интерфейсы.

Android SDK включает некоторые простые виды разметки, которые могут помочь конструировать пользовательские интерфейсы. Только от вас самих зависит выбор правильного сочетания менеджеров компоновки, которые помогут сделать интерфейсы понятными и простыми в использовании.

Перечислим некоторые наиболее универсальные доступные классы разметки.

- FrameLayout. Самый простой из менеджеров компоновки, прикрепляет каждое дочернее Представление к верхнему левому углу экрана. Каждый новый элемент накладывается на предыдущий, заслоняя его.
- LinearLayout. Помещает дочерние Представления в ряд (горизонтальный или вертикальный). Вертикальная разметка представляет собой колонку с элементами, горизонтальная вытягивает их в строку. LinearLayout позволяет задавать «ширину» каждого дочернего

Представления, благодаря чему можно контролировать их размеры в пределах доступного пространства.

- RelativeLayout. Наиболее гибкий среди стандартных видов разметки, позволяет задавать позицию каждого дочернего Представления относительного других элементов и границ экрана.
- TableLayout. Позволяет размещать Представления с помощью сетки, состоящей из строк и столбцов. При этом столбцы могут либо автоматически растягиваться, либо оставаться постоянной ширины.
- Gallery. Отображает элементы в виде однострочного горизонтального списка, который можно прокручивать.

В документации к Android подробно описаны возможности и свойства каждого класса разметки, поэтому, чтобы не дублировать эту информацию, я рекомендую вам посетить страницу http://developer.android.com/guide/topics/ui/layout-objects.html.

Позже в этой главе вы также узнаете, как создавать составные элементы управления (виджеты, состоящие из нескольких взаимодействующих Представлений), расширяя приведенные выше классы разметки.

Использование разметки

Предпочтительный способ реализации разметки — использование XML-файлов в качестве внешних ресурсов. Любой такой файл должен содержать корневой элемент (узел), который, в свою очередь, может включать столько вложенных экземпляров разметки и Представлений, сколько необходимо для построения любых сложных интерфейсов.

В листинге 4.3 показана простая вертикальная разметка LinearLayout, которая помещает TextView над элементом EditText.

Листинг 4.3. Простая разметка LinearLayout в формате XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Enter Text Below"
  />
  <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Text Goes Here!"
  />
</LinearLayout>
```

Обратите внимание, что для каждого элемента разметки вместо абсолютных значений ширины и высоты в пикселах используются константы wrap_content и fill_parent. Это обеспечивает мощный механизм, благодаря которому экземпляры разметки не зависят от размера и разрешения экрана.

Константа wrap_content задает Представлению минимальный размер, необходимый для отображения содержимого (например, высоту, которая нужна для вывода многострочного текста). Константа fill_parent расширяет Представление, чтобы оно заполнило все доступное пространство внутри родительского элемента (или экрана).

В листинге 4.3 разметка заполняет весь экран, тогда как оба текстовых представления пытаются вытянуться во всю доступную ширину. Их высота ограничена размерами, необходимыми для отображения текста.

Позже в этой главе вы научитесь устанавливать минимальные ширину и высоту для собственных элементов управления, а также получите дальнейшие рекомендации по разработке Представлений, не зависящих от разрешения экрана.

Реализация разметки в виде XML-файлов ведет к отделению уровня представления от логики, содержащейся в элементах управления и Активностях. Это также позволяет создавать разновидности разметки, предназначенные для конкретной аппаратной платформы, и загружать их динамически, без изменений в коде программы.

При необходимости (или по желанию) вы можете реализовать разметку в коде приложения. В таком случае, назначая Представления для разметки, важно не забыть о параметрах LayoutParameters. Применить их можно с помощью метода setLayoutParams или во время вызова addView, как показано в листинге 4.4.

Листинг 4.4. Создание разметки LinearLayout в коде программы

```
LinearLayout ll = new LinearLayout(this);
ll.setOrientation(LinearLayout.VERTICAL);

TextView myTextView = new TextView(this);
EditText myEditText = new EditText(this);

myTextView.setText("Enter Text Below");
myEditText.setText("Text Goes Here!");

int lHeight = LinearLayout.LayoutParams.FILL_PARENT;
int lWidth = LinearLayout.LayoutParams.WRAP_CONTENT;

ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight, lWidth));
ll.addView(myEditText, new LinearLayout.LayoutParams(lHeight, lWidth));
setContentView(ll);
```

Оптимизация разметки

Наполнение Активностей экземплярами разметки довольно ресурсоемкий процесс. Каждая новая вложенная разметка (или Представление) может кардинальным образом повлиять на производительность и целостность приложений.

Рекомендуется использовать как можно более простые менеджеры компоновки. Избегайте создания полностью новой разметки только лишь для того, чтобы внести небольшие изменения в уже существующую.

Приведем рекомендации и нормативы по созданию эффективной разметки, но имейте в виду, что они не исчерпывающие.

- **Избегайте излишней вложенности.** Не помещайте одну разметку внутрь другой без необходимости. Помещая `LinearLayout` внутрь FrameLayout и присваивая обоим атрибут `FILL_PARENT`, вы не получите ничего, кроме лишних затрат на заполнение. Обращайте внимание на избыточность разметки, особенно при внесении в нее существенных изменений.
- **Старайтесь не использовать слишком много Представлений.** Каждое дополнительное Представление в разметке отнимает время и ресурсы на создание. Разметка никогда не должна содержать более 80 Представлений, иначе на ее заполнение уйдет слишком много времени.
- **Избегайте глубокой вложенности.** Поскольку разметка поддерживает произвольный уровень вложенности, можно легко создавать сложные, глубоко разветвленные иерархии. И хотя нет никаких жестких ограничений на этот счет, рекомендуется ограничивать вложенность максимум десятью уровнями.

Важно, чтобы вы оптимизировали разметку, повышая эффективность и избавляясь от избыточной вложенности.

Помогает в этом Android SDK, который содержит инструмент командной строки layoutopt. Чтобы проанализировать разметку и получить рекомендации по ее исправлению и улучшению, вызовите команду layoutopt, передав ей в качестве параметров имя ресурса, содержащего разметку (или каталог с ресурсами).

Создание новых Представлений

Расширение существующих Представлений, комбинирование составных элементов и создание уникальных виджетов позволяют реализовывать хорошо выглядящие пользовательские интерфейсы, оптимизированные под задачи вашего приложения. Android предоставляет возможность наследовать набор стандартных Представлений или создавать собственные. Вы получаете

полную свободу и можете адаптировать пользовательские интерфейсы, улучшая впечатление от использования своих программ.

ВНИМАНИЕ

При разработке пользовательских интерфейсов важно найти золотую середину между эстетикой и практичностью. Вместе с возможностью создавать собственные нестандартные Представления приходит соблазн полностью переписать все элементы интерфейса с нуля. Подавите это желание. Стандартные Представления знакомы пользователям по другим приложениям и будут обновляться с выходом новых версий платформы. Учитывая небольшие размеры экрана и довольно ограниченное внимание со стороны пользователя, нужно понимать, что выбор стандартных элементов часто может быть более практичным, нежели создание новых.

Выбор наиболее подходящего метода для создания нового Представления зависит от того, какие цели вы преследуете.

- Изменяйте или расширяйте внешний вид и/или поведение существующего элемента, если он уже поддерживает базовую функциональность, которая вам нужна. Переопределяя обработчики событий и метод onDraw, но используя при этом возможности родительского класса, вы можете изменять Представление, не реализовывая заново его функции. К примеру, можно изменить TextView таким образом, чтобы он отображал заданное количество цифр после десятичного разделителя.
- Объединяйте Представления, чтобы создавать независимые, пригодные для многократного использования элементы, которые сочетают функции нескольких взаимосвязанных компонентов. Например, вы можете создать выпадающий список, объединив элементы TextView и Button, при нажатии которых отображается всплывающее Представление ListView.
- Создавайте полностью новые элементы, если вам нужен особенный интерфейс, который невозможно получить при изменении или совмещении уже существующих Представлений.

Изменение существующих Представлений

Android содержит набор Представлений, которые способны удовлетворить большинство потребностей пользовательского интерфейса. Изменяя их, вы избегаете повторной реализации существующего поведения, адаптируя при этом пользовательский интерфейс и функциональность под требования своего приложения.

Чтобы разработать новое Представление, основанное на уже существующем элементе, создайте класс, который его наследует. Этот процесс показан в листинге 4.5.

Листинг 4.5. Расширение элемента TextView

```
import android.content.Context;
import android.util.AttributeSet;
import android.widget.TextView;

public class MyTextView extends TextView {

  public MyTextView (Context context, AttributeSet attrs, int defStyle)
  {
    super(context, attrs, defStyle);
  }

  public MyTextView (Context context) {
    super(context);
  }

  public MyTextView (Context context, AttributeSet attrs) {
    super(context, attrs);
  }
}
```

Чтобы изменить внешний вид или поведение нового Представления, переопределите и расширьте соответствующие обработчики событий.

В следующем дополнении к листингу 4.5 переопределяется метод onDraw, чтобы изменить внешний вид элемента, а также обработчик onKeyDown, чтобы добавить реакцию на нестандартные нажатия клавиш:

```
public class MyTextView extends TextView {

  public MyTextView (Context context, AttributeSet ats, int defStyle) {
    super(context, ats, defStyle);
  }

  public MyTextView (Context context) {
    super(context);
  }

  public MyTextView (Context context, AttributeSet attrs) {
    super(context, attrs);
  }

  @Override
  public void onDraw(Canvas canvas) {
    [ ... Нарисуйте что-либо на Холсте под текстом ... ]

    // Render the text as usual using the TextView base class.
    super.onDraw(canvas);

    [ ... Нарисуйте что-либо на Холсте над текстом ... ]
  }

  @Override
  public boolean onKeyDown(int keyCode, KeyEvent keyEvent) {
```

```
      [ ... Обработайте каким-то образом ... ]
      [ ... нажатия конкретных клавиш ... ]

      // Используйте уже имеющуюся функциональность, реализованную
      // в базовом классе, чтобы реагировать на нажатия клавиш
      return super.onKeyDown(keyCode, keyEvent);
    }
  }
```

Обработчики событий, доступные внутри Представлений, более подробно будут рассмотрены ниже в этой главе.

Изменение приложения To-Do List. Приложение To-Do List из главы 2 использует элементы TextView для отображения каждой строки в списке ListView. Вы можете изменить внешний вид списка, наследовав класс TextView и переопределив метод onDraw.

В этом примере[1] вы создадите новое Представление TodoListItemView, которое будет выводить элементы списка так, как будто они записаны в бумажном блокноте. По завершении измененная программа To-Do List должна выглядеть, как на рис. 4.1.

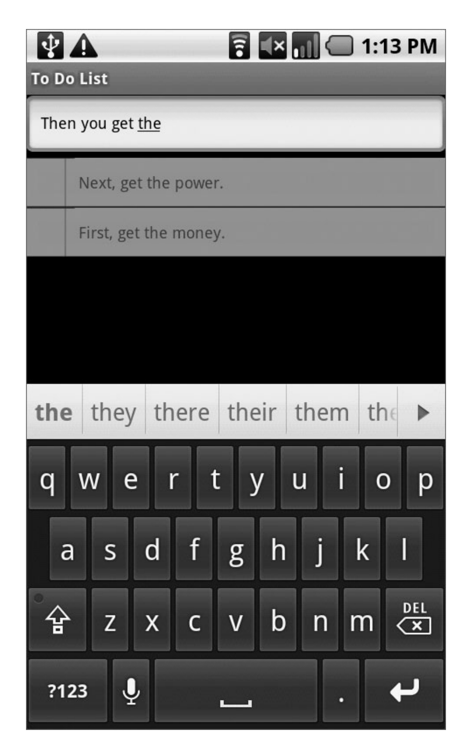


Рис. 4.1.

[1] Все фрагменты кода в этом примере — часть проекта Todo List из главы 4, их можно загрузить с сайта Wrox.com.

1. Создайте новый класс TodoListItemView, который наследует TextView. Добавьте заглушку для переопределения метода onDraw, реализуйте конструктор, в котором вызывается новый метод-заглушка init.

```
package com.paad.todolist;

import android.content.Context;
import android.content.res.Resources;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.widget.TextView;

public class TodoListItemView extends TextView {

  public TodoListItemView (Context context, AttributeSet ats, int ds) {
    super(context, ats, ds);
    init();
  }

  public TodoListItemView (Context context) {
    super(context);
    init();
  }

  public TodoListItemView (Context context, AttributeSet attrs) {
    super(context, attrs);
    init();
  }

  private void init() {
  }

  @Override
  public void onDraw(Canvas canvas) {
    // Используйте родительский класс TextView для вывода текста.
    super.onDraw(canvas);
  }

}
```

2. Создайте новый ресурс `colors.xml` в каталоге **res/values**. Добавьте значения, описывающие цвета бумаги, кромки страницы, линии и текста.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="notepad_paper">#AAFFFF99</color>
  <color name="notepad_lines">#FF0000FF</color>
  <color name="notepad_margin">#90FF0000</color>
  <color name="notepad_text">#AA0000FF</color>
</resources>
```

3. Создайте новый файл `dimens.xml` и добавьте в него значение, описывающее ширину кромки страницы.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <dimen name="notepad_margin">30dp</dimen>
</resources>
```

4. Закончив описание ресурсов, можно начинать изменять внешний вид TodoListItemView. Создайте новые приватные поля для хранения объектов Paint, которые вы будете использовать при рисовании фона и кромки бумажного листа. Также создайте поля для цвета листа и ширины кромки.

   Наполните кодом метод init, добавив в него создание объектов Paint и получение экземпляров тех ресурсов, которые вы создали в двух предыдущих шагах.

```
private Paint marginPaint;
private Paint linePaint;
private int paperColor;
private float margin;

private void init() {
  // Получите ссылку на таблицу ресурсов.
  Resources myResources = getResources();

  // Создайте кисти для рисования, которые мы будем использовать в методе
onDraw.
  marginPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  marginPaint.setColor(myResources.getColor(R.color.notepad_margin));
  linePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  linePaint.setColor(myResources.getColor(R.color.notepad_lines));

  // Получите цвет фона для листа и ширину кромки.
  paperColor = myResources.getColor(R.color.notepad_paper);
  margin = myResources.getDimension(R.dimen.notepad_margin);
}
```

5. Чтобы отобразить лист, переопределите метод onDraw и нарисуйте изображение, используя объекты Paint из пункта 4. Вызовите метод onDraw из родительского класса и позвольте ему нарисовать текст, как он это обычно делает.

```
@Override
public void onDraw(Canvas canvas) {
  // Фоновый цвет для листа
  canvas.drawColor(paperColor);

  // Нарисуйте направляющие линии
  canvas.drawLine(0, 0, getMeasuredHeight(), 0, linePaint);
  canvas.drawLine(0, getMeasuredHeight(),
                  getMeasuredWidth(), getMeasuredHeight(),
                  linePaint);

  // Нарисуйте кромку
```

```
    canvas.drawLine(margin, 0, margin, getMeasuredHeight(), marginPaint);

    // Переместите текст в сторону от кромки
    canvas.save();
    canvas.translate(margin, 0);

    // Используйте TextView для вывода текста.
    super.onDraw(canvas);
    canvas.restore();
  }
```

6. На этом реализация TodoListItemView завершена. Чтобы использовать ее внутри Активности ToDoList, необходимо добавить соответствующий элемент в новую разметку, после чего передать эту разметку в конструктор объекта ArrayAdapter.

   Начните с создания нового ресурса `todolist_item.xml` в каталоге res/layout. В нем указано, каким образом должен отображаться каждый элемент списка. В данном примере разметка должна состоять из единственного элемента TodoListItemView, заполняющего все доступное пространство.

```
<?xml version="1.0" encoding="utf-8"?>
<com.paad.todolist.TodoListItemView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dp"
  android:scrollbars="vertical"
  android:textColor="@color/notepad_text"
  android:fadingEdge="vertical"
/>
```

7. Теперь откройте класс Активности ToDoList. В завершение необходимо изменить параметры, передаваемые в ArrayAdapter внутри метода onCreate. Вместо ссылки, указывающей на стандартный ресурс `android.R.layout.simple_list_item_1`, вставьте ссылку на новую разметку `R.layout.todolist_item`, созданную вами на шаге 6.

```
final ArrayList<String> todoItems = new ArrayList<String>();
int resID = R.layout.todolist_item;
final ArrayAdapter<String> aa = new ArrayAdapter<String>(this, resID,
                                                         todoItems);
myListView.setAdapter(aa);
```

Создание составных элементов управления

Составные элементы управления — это полноценные, пригодные для многократного использования Представления, которые содержат несколько дочерних компонентов, скомпонованных и связанных друг с другом.

При создании составного элемента управления необходимо определить разметку, внешний вид и связи между Представлениями, которые он

содержит. Такие элементы получаются в результате наследования класса ViewGroup (или его производных, например разметки). Чтобы создать новый составной элемент управления, выберите класс разметки, который больше всего подходит для позиционирования дочерних компонентов, и наследуйте его, как показано в листинге 4.6.

Листинг 4.6. Создание составного элемента управления

```
public class MyCompoundView extends LinearLayout {
  public MyCompoundView(Context context) {
    super(context);
  }

  public MyCompoundView(Context context, AttributeSet attrs) {
    super(context, attrs);
  }
}
```

Как и в случае с Активностями, при проектировании разметки для составных Представлений лучше всего использовать внешний ресурс. В листинге 4.7 показан пример описания разметки в формате XML для элемента, содержащего два Представления: поле ввода EditText и кнопку Button, которая это поле очищает.

Листинг 4.7. Ресурс с разметкой для составного Представления

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <EditText
    android:id="@+id/editText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
  <Button
    android:id="@+id/clearButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Clear"
  />
</LinearLayout>
```

Чтобы применить эту разметку, переопределите конструктор нового Представления, добавив в него вызов метода inflate, который принадлежит системному Сервису LayoutInflate. Данный метод принимает в качестве параметра ресурс с разметкой и возвращает «наполненное» Представление.

В случаях, подобных описанному выше, когда возвращаемое Представление должно иметь созданный вами класс, можно передать родительский элемент и автоматически прикрепить к нему результат, как показано ниже.

В листинге 4.8 представлен класс ClearableEditText, в конструкторе которого происходит наполнение ресурса с разметкой, созданного ранее, и получение ссылок на каждое Представление, которое он содержит. При этом вызывается метод hookupButton, используемый для подключения функции «очистки текста», срабатывающей при нажатии кнопки.

Листинг 4.8. Создание составного Представления

```
public class ClearableEditText extends LinearLayout {

  EditText editText;
  Button clearButton;

  public ClearableEditText(Context context) {
    super(context);

    // Наполните Представление из ресурса с разметкой.
    String infService = Context.LAYOUT_INFLATER_SERVICE;
    LayoutInflater li;
    li = (LayoutInflater)getContext().getSystemService(infService);
    li.inflate(R.layout.clearable_edit_text, this, true);

    // Получите ссылки на дочерние элементы.
    editText = (EditText)findViewById(R.id.editText);
    clearButton = (Button)findViewById(R.id.clearButton);

    // Подключите функциональность
    hookupButton();
  }
}
```

Если вы предпочитаете создавать разметку в коде программы, можете сделать это таким же образом, как и в случае с Активностью. В листинге 4.9 показан переопределенный конструктор ClearableEditText, в котором создается пользовательский интерфейс, аналогичный тому, что был описан в формате XML в предыдущем листинге.

Листинг 4.9. Создание разметки для составного Представления в коде программы

```
public ClearableEditText(Context context) {
  super(context);

  // Сделайте разметку вертикальной
  setOrientation(LinearLayout.VERTICAL);

  // Создайте дочерние элементы управления.
  editText = new EditText(getContext());
  clearButton = new Button(getContext());
  clearButton.setText("Clear");

  // Расположите их внутри составного элемента.
  int lHeight = LayoutParams.WRAP_CONTENT;
```

```
    int lWidth = LayoutParams.FILL_PARENT;

    addView(editText, new LinearLayout.LayoutParams(lWidth, lHeight));
    addView(clearButton, new LinearLayout.LayoutParams(lWidth, lHeight));

    // Подключите функциональность
    hookupButton();
  }
```

Создав разметку для Представления, можно подключить обработчики событий для каждого дочернего элемента, чтобы обеспечить нужную функциональность. В представленном ниже фрагменте наполняется кодом метод hookupButton — он отвечает за очистку текста в элементе EditText при нажатии кнопки:

```
  private void hookupButton() {
    clearButton.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View v) {
        editText.setText("");
      }
    });
  }
```

Создание нестандартных Представлений

Создавая полностью новые Представления, вы получаете полный контроль над формированием внешнего вида и поведением своих программ. Можно создавать пользовательские интерфейсы, которые идеально подходят под нужды пользователей. Чтобы выполнить новый элемент с чистого листа, необходимо наследовать один из двух классов — либо View, либо SurfaceView.

Класс View предоставляет объект Canvas, который помимо классов Paint содержит методы для рисования. Используйте их для создания визуального интерфейса на основе растровой графики. Затем вы можете переопределить пользовательские события, такие как нажатия клавиш или касания сенсорного экрана, чтобы обеспечить необходимую интерактивность. Если приложение не должно поддерживать чрезвычайно быструю перерисовку и трехмерную графику, базовый класс View подойдет в качестве мощного и легковесного решения.

Класс SurfaceView предоставляет объект Surface, который поддерживает рисование в фоновом потоке и дает возможность использовать OpenGL для трехмерной графики. Это отличный вариант для насыщенных графикой элементов, которые нуждаются в частых обновлениях или должны отображать сложную графическую информацию, как в случае с играми и трехмерной визуализацией.

Далее вы познакомитесь с двумерными элементами управления, основанными на классе View. Больше информации о классе SurfaceView и о некоторых продвинутых возможностях Canvas, доступных в Android, представлено в главе 15.

Создание нового графического интерфейса

Базовый класс View — это пустая область экрана размером 100 × 100 пикселов. Если нужно изменить размер элемента и сделать так, чтобы он выводил на экран более привлекательный графический интерфейс, необходимо переопределить методы onMeasure и onDraw.

Внутри метода onMeasure новое Представление вычисляет высоту и ширину области, которую оно будет занимать, исходя из заданных условий. В методе onDraw происходит рисование на объекте Canvas.

В листинге 4.10 показан каркас кода для нового класса MyView, который будет рассматриваться и дорабатываться в следующих разделах.

Листинг 4.10. Создание нового класса для Представления

```
public class MyView extends View {

  // Конструктор, необходимый для создания элемента внутри кода программы
  public MyView(Context context) {
    super(context);
  }

  // Конструктор, необходимый для наполнения элемента из файла с ресурсом
  public MyView (Context context, AttributeSet ats, int defaultStyle) {
    super(context, ats, defaultStyle );
  }

  // Конструктор, необходимый для наполнения элемента из файла с ресурсом
  public MyView (Context context, AttributeSet attrs) {
    super(context, attrs);
  }

  @Override
  protected void onMeasure(int wMeasureSpec, int hMeasureSpec) {
    int measuredHeight = measureHeight(hMeasureSpec);
    int measuredWidth = measureWidth(wMeasureSpec);

    // Вы ДОЛЖНЫ сделать вызов метода setMeasuredDimension,
    // иначе получится выброс исключения при
    // размещении элемента внутри разметки.
    setMeasuredDimension(measuredHeight, measuredWidth);
  }

  private int measureHeight(int measureSpec) {
    int specMode = MeasureSpec.getMode(measureSpec);
    int specSize = MeasureSpec.getSize(measureSpec);

    [ ... Вычисление высоты Представления ... ]

    return specSize;
  }

  private int measureWidth(int measureSpec) {
    int specMode = MeasureSpec.getMode(measureSpec);
```

```
      int specSize = MeasureSpec.getSize(measureSpec);

      [ ... Вычисление ширины Представления  ... ]

      return specSize;
    }

    @Override
    protected void onDraw(Canvas canvas) {
      [ ... Рисуйте здесь ваш графический интерфейс ... ]
    }
  }
```

ВНИМАНИЕ

Внутри обработчика onMeasure вызывается метод setMeasuredDimension. Вы должны всегда выполнять это действие, иначе ваш элемент выбросит исключение, когда родительский контейнер попытается разместить его на экране.

Отрисовка вашего элемента управления

Именно в методе onDraw происходит вся «магия». Создавая новый виджет с нуля, вы, вероятно, хотите спроектировать для него совершенно новый графический интерфейс. Параметр Canvas в методе onDraw представляет собой Холст (Canvas), который используется для реализации идей.

Android предоставляет инструменты, с помощью которых можно изобразить свои задумки, используя объекты Canvas и Paint. Класс Canvas содержит вспомогательные методы для рисования простых двумерных примитивов, включая окружности, линии, прямоугольники, текст и объекты Drawable (изображения). Кроме того, он поддерживает преобразования, с помощью которых вы сможете вращать, транслировать (передвигать) и масштабировать объект Canvas, рисуя на нем.

При использовании этих инструментов в сочетании с классами Drawable и Paint (последний предоставляет ряд настраиваемых заполнителей и контуров) сложность и детализация вашего элемента ограничиваются только размерами экрана и мощностью процессора, который все это рисует.

ВНИМАНИЕ

Один из наиболее важных моментов при написании эффективного кода в Android — необходимость избегать повторного создания и разрушения объектов. Любой объект, выполненный внутри метода onDraw, будет создаваться и разрушаться при каждом обновлении экрана. Повышайте эффективность своего кода, перенося как можно больше таких объектов (в частности, экземпляры Paint и Drawable) в область видимости класса, передавая их создание конструктору.

В листинге 4.11 показано, как переопределить метод onDraw, чтобы отображать в центре экрана простую текстовую строку.

Листинг 4.11. Отрисовка нестандартного Представления

```
@Override
protected void onDraw(Canvas canvas) {
  // Получите размер элемента, основываясь на последнем вызове
обработчика onMeasure.
  int height = getMeasuredHeight();
  int width = getMeasuredWidth();

  // Найдите центр
  int px = width/2;
  int py = height/2;

  // Создайте новые кисти для рисования.
  // ЗАМЕТКА: В целях повышения эффективности, это
  // должно быть сделано в конструкторе Представления
  Paint mTextPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  mTextPaint.setColor(Color.WHITE);

  // Определите строку.
  String displayText = "Hello World!";

  // Измерьте ширину текстовой строки.
  float textWidth = mTextPaint.measureText(displayText);

  // Нарисуйте текстовую строку в центре элемента.
  canvas.drawText(displayText, px-textWidth/2, py, mTextPaint);
}
```

Более подробная информация о методах рисования сложных графических элементов содержится в главе 15.

ПРИМЕЧАНИЕ

На сегодняшний день Android не имеет поддержки векторной графики. Как результат, изменения в любом элементе, размещенном на объекте Canvas, приводят к перерисовке всего Холста, изменение цвета кисти не повлияет на отображаемое Представление, пока оно не будет помечено как недействительное и не перерисуется. В качестве альтернативы для отрисовки графики вы можете использовать OpenGL. Для получения более подробной информации ознакомьтесь с описанием класса SurfaceView в главе 15.

Задание размеров для вашего элемента

Если ваш элемент не может идеально вписаться в площадь 100 × 100 пикселов, вам необходимо переопределить обработчик onMeasure.

Данный метод вызывается в момент, когда родительское Представление размещает внутри себя дочерние элементы. Оно как бы спрашивает: «Сколько места вы собираетесь использовать?» — и передает два параметра: widthMeasureSpec и heightMeasureSpec. Они определяют доступное для элемента пространство, а также некоторые метаданные, описывающие его.

Вместо того чтобы возвращать результат, необходимо передать высоту и ширину Представления в метод setMeasuredDimension.

В листинге 4.12 показано, как переопределить обработчик onMeasure. Обратите внимание на вызовы локальных методов-заглушек calculateHeight и calculateWidth. Они будут использоваться для декодирования значений widthHeightSpec и heightMeasureSpec, а также для вычисления предпочтительных высоты и ширины.

Листинг 4.12. Определение размеров Представления

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

  int measuredHeight = measureHeight(heightMeasureSpec);
  int measuredWidth = measureWidth(widthMeasureSpec);

  setMeasuredDimension(measuredHeight, measuredWidth);
}

private int measureHeight(int measureSpec) {
  // Верните измеренную высоту виджета.
}

private int measureWidth(int measureSpec) {
  // Верните измеренную ширину виджета.
}
```

Параметры widthMeasureSpec и heightMeasureSpec, описывающие границы элемента, из соображений эффективности передаются в виде целочисленных значений. Прежде чем их использовать, они должны быть декодированы с помощью статических методов getMode и getSize, принадлежащих классу MeasureSpec:

```
int specMode = MeasureSpec.getMode(measureSpec);
int specSize = MeasureSpec.getSize(measureSpec);
```

В зависимости от режима mode, свойство size представляет собой либо максимальное пространство, доступное для элемента управления (при режиме AT_MOST), либо точные размеры области, которую займет Представление (при режиме EXACTLY). Если режим был установлен в UNSPECIFIED, ваш элемент не будет знать, какое именно значение описывает свойство size.

Задавая свойству mode значение EXACTLY, родительский элемент декларирует, что Представление будет помещено именно в ту область, размер которой был указан. В режиме AT_MOST родительский элемент спрашивает Представление, какой размер оно хочет занять, предлагая при этом определенные границы, за которые оно не может выйти. Часто бывает, что в обоих случаях возвращаемые значения идентичны.

Как бы то ни было, вы должны рассматривать эти ограничения как абсолютные. Иногда целесообразно возвращать размеры, выходящие за эти

рамки. В таких случаях родительский элемент сам выбирает, как поступать с Представлением, используя обрезание или прокрутку.

В листинге 4.13 показана типичная реализация управления размерами Представления.

Листинг 4.13. Типичная реализация управления размерами Представления

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
  int measuredHeight = measureHeight(heightMeasureSpec);
  int measuredWidth = measureWidth(widthMeasureSpec);

  setMeasuredDimension(measuredHeight, measuredWidth);
}

private int measureHeight(int measureSpec) {
  int specMode = MeasureSpec.getMode(measureSpec);
  int specSize = MeasureSpec.getSize(measureSpec);

  // Размер по умолчанию, если ограничения не были установлены.
  int result = 500;

  if (specMode == MeasureSpec.AT_MOST) {
    // Рассчитайте идеальный размер вашего
    // элемента в рамках максимальных значений.
    // Если ваш элемент заполняет все доступное
    // пространство, верните внешнюю границу.
    result = specSize;
  } else if (specMode == MeasureSpec.EXACTLY) {
    // Если ваш элемент может поместиться внутри этих границ, верните это
значение.
    result = specSize;
  }
  return result;
}

private int measureWidth(int measureSpec) {
  int specMode = MeasureSpec.getMode(measureSpec);
  int specSize = MeasureSpec.getSize(measureSpec);

  // Размер по умолчанию, если ограничения не были установлены.
  int result = 500;

  if (specMode == MeasureSpec.AT_MOST) {
    // Рассчитайте идеальный размер вашего
    // элемента в рамках максимальных значений.
    // Если ваш элемент заполняет все доступное
    // пространство, верните внешнюю границу.
    result = specSize;
  } else if (specMode == MeasureSpec.EXACTLY) {
    // Если ваш элемент может поместиться внутри этих границ, верните это
значение.
    result = specSize;
  }
  return result;
}
```

Обработка событий, основанных на взаимодействии с пользователем

Чтобы новое Представление было интерактивным, оно должно реагировать на нажатие клавиш и касание экрана. В Android есть несколько обработчиков событий, которые позволяют реагировать на пользовательский ввод.

- onKeyDown. Вызывается при нажатии любой аппаратной клавиши, включая манипулятор D-pad, клавиатуру, а также кнопки для телефонного вызова, отмены звонка, возврата и управления камерой.
- onKeyUp. Вызывается, когда пользователь отпускает нажатую клавишу.
- onTrackballEvent. Вызывается при перемещении трекбола.
- onTouchEvent. Вызывается при нажатии/отпускании сенсорного экрана или же при обнаружении движения.

В листинге 4.14 показан каркас класса, в котором переопределяются все обработчики событий, отвечающие за взаимодействие с пользователем.

Листинг 4.14. Обработка пользовательского ввода для Представления

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent keyEvent) {
  // Верните значение true, если событие было обработано.
  return true;
}

@Override
public boolean onKeyUp(int keyCode, KeyEvent keyEvent) {
  // Верните значение true, если событие было обработано.
  return true;
}

@Override
public boolean onTrackballEvent(MotionEvent event ) {
  // Получите тип действия, которое представлено данным событием.
  int actionPerformed = event.getAction();
  // Верните значение true, если событие было обработано.
  return true;
}

@Override
public boolean onTouchEvent(MotionEvent event) {
  // Получите тип действия, которое представлено данным событием.
  int actionPerformed = event.getAction();
  // Верните значение true, если событие было обработано.
  return true;
}
```

Подробности об использовании каждого из этих обработчиков, включая детальную информацию о параметрах, принимаемых данными методами, а также сведения о поддержке множественных касаний представлены в главе 15.

Создание приложения Compass View

В следующем примере[1] вы создадите новое Представление CompassView, наследуя класс View. Этот элемент будет выводить на экран традиционное изображение компаса (в виде розы ветров), указывая курс (направление). По завершении он должен выглядеть так, как показано на рис. 4.2.

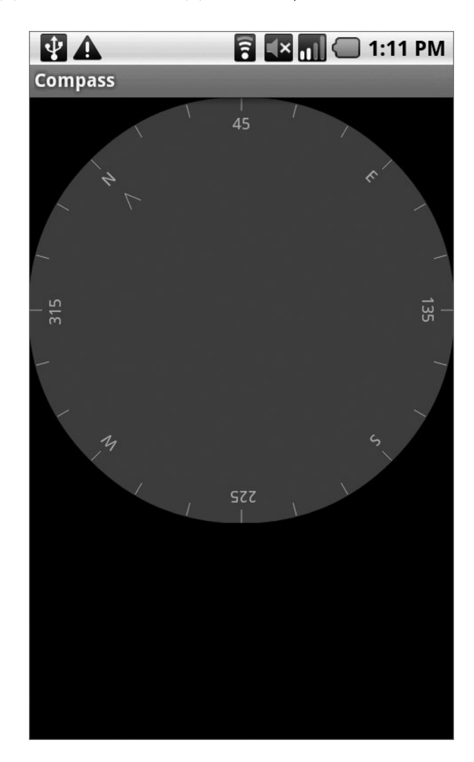


Рис. 4.2.

Этот компас — пример графического Представления, которое требует абсолютно иного подхода к отображению, нежели элементы TextView и Button, доступные в SDK. Это делает его отличным кандидатом на создание с нуля.

ПРИМЕЧАНИЕ

В главе 14 вы будете использовать Представление CompassView совместно с аппаратным акселерометром, чтобы отображать текущее направление пользователя. В главе 15 изучите продвинутые методики для рисования на объекте Canvas, с помощью которых можно кардинальным образом улучшить внешний вид этого компаса.

[1] Все фрагменты кода в этом примере — часть проекта Compass из главы 4, их можно загрузить с сайта Wrox.com.

1. Создайте новый проект под названием Compass, а также Активность для вывода на экран вашего нового Представления CompassView. Затем выполните новый класс CompassView, наследующий View. Создайте конструкторы, которые позволят получать экземпляры Представления как внутри кода программы, так и с помощью ресурса с разметкой. Добавьте новый метод initCompassView — он будет использоваться для инициализации элемента — и вызовите его внутри каждого конструктора.

```
package com.paad.compass;

import android.content.Context;
import android.graphics.*;
import android.graphics.drawable.*;
import android.view.*;
import android.util.AttributeSet;
import android.content.res.Resources;

public class CompassView extends View {
  public CompassView(Context context) {
    super(context);
    initCompassView();
  }

  public CompassView(Context context, AttributeSet attrs) {
    super(context, attrs);
    initCompassView();
  }

  public CompassView(Context context,
                     AttributeSet ats,
                     int defaultStyle) {
    super(context, ats, defaultStyle);
    initCompassView();
  }

  protected void initCompassView() {
    setFocusable(true);
  }
}
```

2. Элемент, отображающий компас, всегда должен представлять собой идеальную окружность, занимающую все доступное пространство Холста. Переопределите обработчик onMeasure, чтобы вычислить длину короткой грани, и вызовите метод setMeasuredDimension, чтобы установить высоту и ширину, используя полученное значение.

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
  // Компас представляет собой окружность, занимающую все доступное
  // пространство.
  // Установите размеры элемента, вычислив короткую грань (высоту
  // или ширину).
  int measuredWidth = measure(widthMeasureSpec);
```

```
    int measuredHeight = measure(heightMeasureSpec);

    int d = Math.min(measuredWidth, measuredHeight);

    setMeasuredDimension(d, d);
  }

  private int measure(int measureSpec) {
    int result = 0;

    // Декодируйте параметр measureSpec.
    int specMode = MeasureSpec.getMode(measureSpec);
    int specSize = MeasureSpec.getSize(measureSpec);

    if (specMode == MeasureSpec.UNSPECIFIED) {
      // Если границы не указаны, верните размер по умолчанию (200).
      result = 200;
    } else {
      // Так как вам нужно заполнить все доступное пространство,
      // всегда возвращайте максимальный доступный размер.
      result = specSize;
    }
    return result;
  }
```

3. Создайте два новых файла с ресурсами для хранения цветов и текстовых строк, которые будут использоваться при рисовании компаса.

   3.1. Создайте ресурс со строками `res/values/strings.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Compass</string>
  <string name="cardinal_north">N</string>
  <string name="cardinal_east">E</string>
  <string name="cardinal_south">S</string>
  <string name="cardinal_west">W</string>
</resources>
```

   3.2. Создайте ресурс со значениями цветов `res/values/colors.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="background_color">#F555</color>
  <color name="marker_color">#AFFF</color>
  <color name="text_color">#AFFF</color>
</resources>
```

4. Теперь вернитесь к классу CompassView. Добавьте новое пустое свойство, чтобы хранить отображаемое направление, и создайте для него геттер и сеттер.

```
private float bearing;

public void setBearing(float _bearing) {
  bearing = _bearing;
```

```
}
public float getBearing() {
  return bearing;
}
```

5. Перейдите к методу initCompassView и получите ссылки на ресурсы, созданные на шаге 3. Сохраните строковые значения в виде свойств, используйте значения цветов для создания нового объекта Paint, принадлежащего классу. Эти объекты вы примените позже при рисовании циферблата для компаса.

```
private Paint markerPaint;
private Paint textPaint;
private Paint circlePaint;
private String northString;
private String eastString;
private String southString;
private String westString;
private int textHeight;

protected void initCompassView() {
  setFocusable(true);

  circlePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  circlePaint.setColor(r.getColor(R.color.background_color));
  circlePaint.setStrokeWidth(1);
  circlePaint.setStyle(Paint.Style.FILL_AND_STROKE);

  Resources r = this.getResources();
  northString = r.getString(R.string.cardinal_north);
  eastString = r.getString(R.string.cardinal_east);
  southString = r.getString(R.string.cardinal_south);
  westString = r.getString(R.string.cardinal_west);

  textPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  textPaint.setColor(r.getColor(R.color.text_color));

  textHeight = (int)textPaint.measureText("yY");

  markerPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  markerPaint.setColor(r.getColor(R.color.marker_color));
}
```

6. В завершение необходимо нарисовать циферблат компаса, используя объекты String и Paint, созданные в пункте 5. Следующий фрагмент кода сопровождается всего лишь краткими комментариями. В главе 15 вы можете найти больше подробностей о рисовании на объекте Canvas с использованием продвинутых эффектов.

   6.1. В первую очередь начните с того, что переопределите метод onDraw в классе CompassView.

```
@Override
protected void onDraw(Canvas canvas) {
```

6.2. Найдите центральную точку элемента управления и сохраните короткую грань в качестве радиуса компаса.

```
int px = getMeasuredWidth() / 2;
int py = getMeasuredHeight() /2 ;

int radius = Math.min(px, py);
```

6.3. С помощью метода drawCircle нарисуйте внешнюю границу и добавьте фон для циферблата. Воспользуйтесь объектом circlePaint, который создали в пункте 5.

```
// Нарисуйте фон
canvas.drawCircle(px, py, radius, circlePaint);
```

6.4. Компас отображает текущее направление таким образом, чтобы оно всегда указывало на верхнюю часть устройства, поворачивая при этом циферблат. Чтобы достичь такого эффекта, поворачивайте `Холст` в направлении, противоположном текущему.

```
// Поворачивайте ракурс таким образом, чтобы
// "верх" всегда указывал на текущее направление.
canvas.save();
canvas.rotate(-bearing, px, py);
```

6.5. Все, что теперь осталось сделать, — нарисовать метки. Сделайте полный поворот `Холста`, рисуя отметки каждые 15° и обозначая направления каждые 45°.

```
int textWidth = (int)textPaint.measureText("W");
int cardinalX = px-textWidth/2;
int cardinalY = py-radius+textHeight;

// Рисуйте отметки каждые 15° и текст каждые 45°.
for (int i = 0; i < 24; i++) {
  // Нарисуйте метку.
  canvas.drawLine(px, py-radius, px, py-radius+10, markerPaint);

  canvas.save();
  canvas.translate(0, textHeight);

  // Нарисуйте основные точки
  if (i % 6 == 0) {
    String dirString = "";
    switch (i) {
      case(0) : {
                  dirString = northString;
                  int arrowY = 2*textHeight;
                  canvas.drawLine(px, arrowY, px-5, 3*textHeight,
                                  markerPaint);
                  canvas.drawLine(px, arrowY, px+5, 3*textHeight,
                                  markerPaint);
                  break;
                }
      case(6)  : dirString = eastString; break;
      case(12) : dirString = southString; break;
      case(18) : dirString = westString; break;
```

```
      }
      canvas.drawText(dirString, cardinalX, cardinalY, textPaint);
    }

    else if (i % 3 == 0) {
      // Отображайте текст каждые 45°
      String angle = String.valueOf(i*15);
      float angleTextWidth = textPaint.measureText(angle);

      int angleTextX = (int)(px-angleTextWidth/2);
      int angleTextY = py-radius+textHeight;
      canvas.drawText(angle, angleTextX, angleTextY, textPaint);
    }
    canvas.restore();

    canvas.rotate(15, px, py);
  }
  canvas.restore();
}
```

7. Чтобы вывести компас на экран, отредактируйте ресурс с разметкой `main.xml`, заменив TextView на новосозданный элемент CompassView. Этот процесс более подробно описан в следующем разделе.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <com.paad.compass.CompassView
    android:id="@+id/compassView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
  />
</LinearLayout>
```

8. Запустив Активность, вы должны увидеть на экране CompassView. Чтобы узнать, как связать CompassView с аппаратным компасом в устройстве, перейдите к главе 14.

Использование нестандартных элементов управления

Создавая собственные Представления, вы можете использовать их внутри кода программы и в разметке, как и любые другие элементы. В листинге 4.15 показано, как переопределить метод onCreate, чтобы добавить в Активность элемент CompassView, созданный в предыдущем примере.

Листинг 4.15. Использование нестандартного Представления

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  CompassView cv = new CompassView(this);
  setContentView(cv);
  cv.setBearing(45);
}
```

Чтобы использовать этот же элемент внутри ресурса, укажите полное имя класса при создании нового узла в описании разметки, как показано в следующем фрагменте XML-кода:

```
<com.paad.compass.CompassView
  android:id="@+id/compassView"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
/>
```

Вы можете наполнить разметку и получить ссылку на CompassView как обычно, используя следующий код:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  CompassView cv = (CompassView)this.findViewById(R.id.compassView);
  cv.setBearing(45);
}
```

Ресурсы Drawable

В главе 3 вы познакомились с системой ресурсов, узнали, как отделять ресурсы от программы и подключать их версии для разных аппаратных платформ.

В этом разделе вы познакомитесь с несколькими новыми типами ресурсов для рисования, включая фигуры, преобразования и компоновку, узнаете, каким образом их можно применять при создании пользовательских интерфейсов, не зависящих от размера и разрешения экрана.

Все эти ресурсы могут быть описаны и использованы в коде программы, но в этом разделе мы будем создавать их с помощью XML.

ПРИМЕЧАНИЕ

Фреймворк, о котором шла речь в главе 3, годится не только для определения альтернативных ресурсов, нацеленных на разные аппаратные конфигурации, но может быть использован и при описании ресурсов Drawable, рассматриваемых в этом разделе.

Фигуры, цвета и градиенты

Android включает простые ресурсы для рисования, которые можно полностью описать в формате XML. Это касается классов ColorDrawable, ShapeDrawable и GradientDrawable. Данные ресурсы хранятся в каталоге res/drawable и могут быть идентифицированы в коде приложения по именам файлов, записанным в нижнем регистре.

Если описывать эти ресурсы в формате XML и указывать атрибуты для них с помощью аппаратно-независимых пикселов (density-independent pixels), система сможет их плавно масштабировать. Как и в случае с векторной графикой, эти ресурсы могут динамически масштабироваться, отображаясь корректно и без артефактов при любых размерах и разрешениях экрана, независимо от плотности пикселов. Исключение — ресурс GradientDrawable, радиус для которого должен быть указан в пикселах.

По ходу чтения данной главы вы увидите, что эти элементы могут быть использованы в сочетании с ресурсами для трансформации и компоновки. Комбинируя их, вы сможете создавать динамические, легковесные и масштабируемые элементы пользовательского интерфейса, выглядящие одинаково четко на любом экране.

ColorDrawable

ColorDrawable — простейший ресурс для рисования, он позволяет указывать свойство изображения, основанное на единственном сплошном цвете. ColorDrawable описывается в виде XML-файлов (хранящихся в каталоге с ресурсами) с помощью тега `<color>`. В листинге 4.16 показан код для ресурса, описывающего сплошной красный цвет.

Листинг 4.16. Ресурс, описывающий сплошной красный цвет

```
<color xmlns:android="http://schemas.android.com/apk/res/android"
  android:color="#FF0000"
/>
```

ShapeDrawable

Данный вид ресурсов позволяет описывать простые геометрические фигуры, указывая их размеры, фон и контур с помощью тега `<shape>`.

Каждый такой тег состоит из типа (указывается с помощью атрибута `android:shape`), атрибутов, определяющих размер фигуры, и дочерних узлов, в которых задаются значения для отступов, контура (или очертания) и фона.

На сегодняшний день Android поддерживает несколько типов фигур, задать которые можно в атрибуте `android:shape`.

- oval. Простой овал.
- rectangle. Поддерживает также вложенный тег `<corners>`, с помощью которого можно создать прямоугольник с закругленными углами (используя атрибут `radius`).
- ring (кольцо). Поддерживает атрибуты `innerRadius` и `thickness`, с помощью которых задаются внутренний радиус кольца и его толщина соответственно. В качестве альтернативы вы можете использовать

атрибуты innerRadiusRatio и/или thicknessRatio, чтобы указать те же параметры в виде значений, пропорциональных ширине (где внутренний радиус, равный четверти ширины, будет использовать значение 4).

Используйте вложенный тег <stroke>, чтобы с помощью атрибутов width и color задать контур для своих фигур.

Вы также можете добавить узел <padding>, чтобы задать отступ при позиционировании вашей фигуры на Холсте.

Что еще более полезно — можно включить в описание дочерний тег для определения фонового цвета. В простейшем случае это использование узла <solid> в сочетании с атрибутом color, который описывает сплошной цвет заливки.

Следующий раздел посвящен классу GradientDrawable. Вы узнаете, каким образом можно задать градиентную заливку для различных типов ShapeDrawable.

В листинге 4.17 показан ресурс с фигурой (прямоугольником), которая имеет сплошной фон, закругленные углы, отступ от каждой грани размером 10 dp и такой же толщины контур. Результат представлен на рис. 4.3.

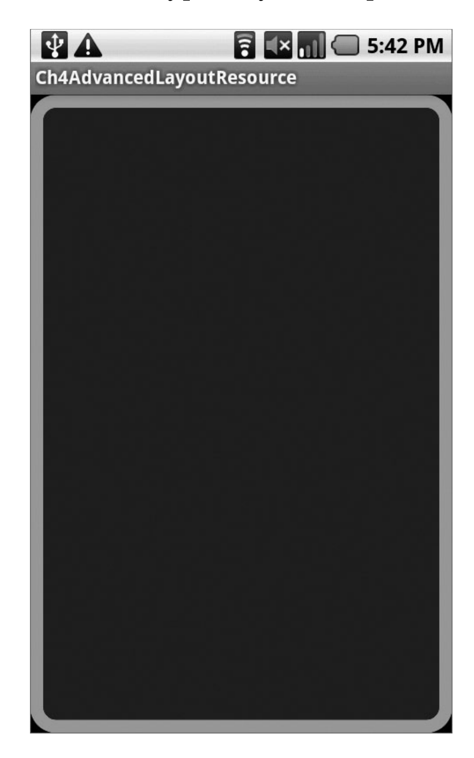


Рис. 4.3.

Листинг 4.17. Ресурс для рисования, залитый сплошным красным цветом

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle">
  <solid
    android:color="#f0600000"/>
  <stroke
    android:width="10dp"
    android:color="#00FF00"/>
  <corners
    android:radius="15dp" />
  <padding
    android:left="10dp"
    android:top="10dp"
    android:right="10dp"
    android:bottom="10dp"
  />
</shape>
```

GradientDrawable

GradientDrawable позволяет создавать сложные градиентные заливки. Каждый градиент описывает плавный переход между двумя или тремя цветами с помощью линейного/радиального алгоритма или же используя метод развертки.

GradientDrawable описывается в виде тега `<gradient>`, находясь внутри определения ресурса ShapeDrawable (см. выше).

Каждый ресурс с градиентом должен содержать как минимум по одному атрибуту `startColor` и `endColor`. Атрибут `middleColor` необязателен. Используя атрибут `type`, вы можете описать свой градиент.

- linear. Стандартный тип градиента. Отображает прямой переход от цвета startColor к цвету endColor под углом, заданным в атрибуте `angle`.
- radial. Рисует круговой градиент, начиная с цвета startColor и заканчивая startColor, от внешнего края фигуры до ее центра. Требует атрибут `gradientRadius`, который указывает радиус градиентного перехода в пикселах. Поддерживаются также необязательные атрибуты `centerX` и `centerY`, описывающие сдвиг центральной точки градиента.

  Поскольку радиус градиента указывается в пикселах, он не будет автоматически масштабироваться при разной плотности точек на экране. Чтобы минимизировать эффект ступенчатости, необходимо указывать разные значения радиуса для дисплеев с разным разрешением.
- sweep. Рисует разверточный градиент с помощью перехода между цветами startColor и endColor вдоль внешнего края фигуры (как правило, кольца).

В листинге 4.18 показан ресурс в формате XML, описывающий линейный градиент внутри прямоугольника, радиальный градиент внутри овала, а также разверточный градиент внутри кольца. Результат можно увидеть на рис. 4.4.

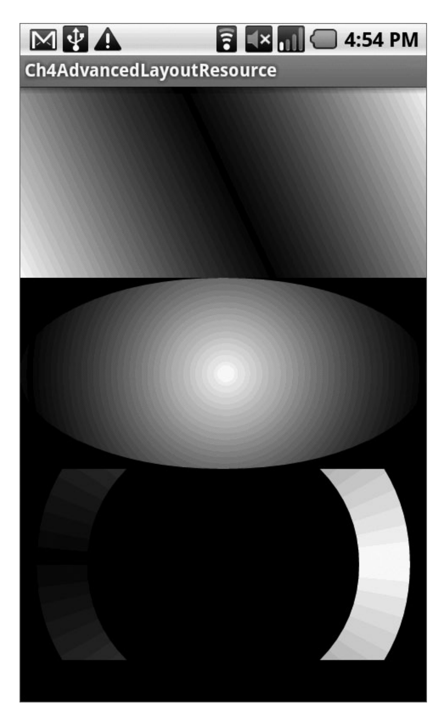


Рис. 4.4.

Листинг 4.18. Описание линейного, радиального и разверточного градиентов

```
<!-- Прямоугольник с линейным градиентом -->
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle"
  android:useLevel="false">
  <gradient
    android:startColor="#ffffff"
    android:endColor="#ffffff"
    android:centerColor="#000000"
    android:useLevel="false"
    android:type="linear"
    android:angle="45"
  />
</shape>

<!-- Овал с радиальным градиентом -->
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="oval"
  android:useLevel="false">
```

```
  <gradient
    android:type="radial"
    android:startColor="#ffffff"
    android:endColor="#ffffff"
    android:centerColor="#000000"
    android:useLevel="false"
    android:gradientRadius="300"
  />
</shape>

<!-- Кольцо с разверточным градиентом -->
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="ring"
  android:useLevel="false"
  android:innerRadiusRatio="3"
  android:thicknessRatio="8">
  <gradient
    android:startColor="#ffffff"
    android:endColor="#ffffff"
    android:centerColor="#000000"
    android:useLevel="false"
    android:type="sweep"
/>
</shape>
```

Композитные ресурсы для рисования

Используйте данный вид ресурсов для объединения и манипулирования другими ресурсами.

Внутри описания композитных ресурсов могут быть использованы растровые изображения, геометрические фигуры и цвета. Кроме того, сами композитные ресурсы можно использовать один внутри другого, назначая их для Представлений точно так же, как и любые другие виды ресурсов.

Ресурсы, описывающие преобразования

Вы можете масштабировать и поворачивать ресурсы Drawable, используя классы ScaleDrawable и RotateDrawable. Такие преобразования чрезвычайно полезны при создании индикаторов выполнения задач или для добавления анимации к Представлениям.

- ScaleDrawable. Используйте атрибуты `scaleHeight` и `scaleWidth` внутри тега `<scale>`, чтобы описать высоту и ширину относительно границ оригинального объекта Drawable. Добавьте атрибут `scaleGravity` для изменения опорной точки масштабированного изображения.
- RotateDrawable. Используйте атрибуты `fromDegrees` и `toDegrees` внутри тега `<rotate>`, чтобы задать начальный и конечный углы поворота вокруг опорной точки. Опорная точка указывается с помощью атрибутов `pivotX` и `pivotY`, которые содержат процентные значения относительно ширины и высоты объекта Drawable соответственно. Запись делается в виде `nn%`.

Чтобы применить масштабирование и поворот в процессе выполнения программы, используйте метод setLevel из Представления, содержащего объект Drawable, переходя при этом от начального к конечному значению (от 0 до 10 000).

В процессе перехода уровень 0 представляет начальный угол (или наименьший масштаб). Уровень 10 000 указывает на завершение преобразования (конечный угол или самый большой масштаб).

В листинге 4.19 описаны ресурсы ScaleDrawable и RotateDrawable. В листинге 4.20 показано, как управлять этими ресурсами в коде программы после того, как они были назначены для объекта ImageView.

Листинг 4.19. Файлы с ресурсами RotateDrawable и ScaleDrawable

```
<!-- Ресурс RotationDrawable -->
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
  android:drawable="@drawable/icon"
  android:fromDegrees="0"
  android:toDegrees="90"
  android:pivotX="50%"
  android:pivotY="50%"
/>

<!-- Ресурс ScaleDrawable -->
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
  android:drawable="@drawable/icon"
  android:scaleHeight="100%"
  android:scaleWidth="100%"
/>
```

Листинг 4.20. Применение преобразований (поворота и масштабирования) в коде программы

```
ImageView rotatingImage = (ImageView)findViewById(R.
id.RotatingImageView);
ImageView scalingImage = (ImageView)findViewById(R.id.ScalingImageView);

// Поверните изображение на 50% от итогового угла.
rotatingImage.setImageLevel(5000);

// Масштабируйте изображение до 50% от его финального размера.
scalingImage.setImageLevel(5000);
```

LayerDrawable

LayerDrawable позволяет накладывать несколько объектов Drawable один поверх другого. Описав массив полупрозрачных объектов Drawable,

вы можете создать сложную комбинацию динамических фигур и преобразований.

Кроме того, вы можете применять к LayerDrawable те же преобразования, которые были описаны в предыдущем разделе, а также сочетать их с ресурсами StateListDrawable и LevelListDrawable, речь о которых пойдет дальше.

В листинге 4.21 LayerDrawable описывается с помощью тега `<layer-list>`, внутри которого для каждого дочернего узла `<item>` используется атрибут `drawable`, указывающий на ресурс для наложения.

Каждый объект Drawable будет накладываться в соответствии со своим индексом — первый элемент массива размещается в самом низу.

Листинг 4.21. Описание ресурса LayerDrawable в формате XML

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/bottomimage"/>
  <item android:drawable="@drawable/image2"/>
  <item android:drawable="@drawable/image3"/>
  <item android:drawable="@drawable/topimage"/>
</layer-list>
```

StateListDrawable

StateListDrawable — это композитный ресурс, позволяющий отображать разные объекты Drawable в зависимости от состояния Представления, для которого они были назначены.

Ресурс StateListDrawable используется в большинстве стандартных Представлений в Android, например в виде изображений кнопок или фона для обычных элементов списка ListView.

Чтобы описать StateListDrawable, создайте файл в формате XML, в котором указываются разные ресурсы Drawable для каждого состояния Представления, как показано в листинге 4.22.

Листинг 4.22. Ресурс StateListDrawable

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_window_focused="false"
        android:drawable="@drawable/widget_bg_normal"/>
  <item android:state_pressed="true"
        android:drawable="@drawable/widget_bg_pressed"/>
  <item android:state_focused="true"
        android:drawable="@drawable/widget_bg_selected"/>
  <item android:drawable="@drawable/widget_bg_normal"/>
</selector>
```

LevelListDrawable

Используя LevelListDrawable, вы можете эффективно размещать ресурсы Drawable один поверх другого, указывая целочисленный индекс для каждого слоя, как показано в листинге 4.23.

Листинг 4.23. Ресурс LevelListDrawable

```
<level-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:maxLevel="0" android:drawable="@drawable/earthquake_0"/>
  <item android:maxLevel="1" android:drawable="@drawable/earthquake_1"/>
  <item android:maxLevel="2" android:drawable="@drawable/earthquake_2"/>
  <item android:maxLevel="4" android:drawable="@drawable/earthquake_4"/>
  <item android:maxLevel="6" android:drawable="@drawable/earthquake_6"/>
  <item android:maxLevel="8" android:drawable="@drawable/earthquake_8"/>
  <item android:maxLevel="10" android:drawable="@drawable/earthquake_10"/>
</level-list>
```

Чтобы вывести на экран определенное изображение, вызовите метод setImageLevel из Представления, которому назначен ресурс LevelListDrawable, передавая в качестве параметра индекс объекта Drawable, который вы хотите отобразить.

```
imageView.setImageLevel(5);
```

Представление отобразит ресурс с соответствующим (или большим) индексом. Ресурс LevelListDrawable нужен при компоновке виджетов.

Ресурс NinePatch

Изображения формата NinePatch (растягивающиеся) — это PNG-файлы, где некоторые помеченные части могут быть растянуты. Они должны описываться в формате PNG и иметь расширение `.9.png`. Идентификатор ресурсов для NinePatch — это имя файла без окончания `.9.png`.

NinePatch — разновидность PNG-файлов, которые содержат рамку толщиной в один пиксел. Данная рамка описывает область, которая может растягиваться при изменении размеров изображения. Чтобы создать ресурс NinePatch, нарисуйте по его левому и верхнему краям изображения черные полоски толщиной в один пиксел, описывающие растягиваемый участок. Части рисунка, не отмеченные подобным образом, останутся прежними; изменения размеров изображения будут пропорционально влиять на относительные размеры помеченных участков.

ПРИМЕЧАНИЕ

Ресурс NinePatch — мощный инструмент, способный создавать фоновые изображения для Представлений или Активностей, размеры которых могут меняться. К примеру, NinePatch используется в Android при создании обрамления для кнопок.

Интерфейсы, не зависящие от разрешения и плотности пикселов

Поскольку первые четыре мобильных телефона под управлением Android имели экраны размером 3,2″ и разрешение HVGA, разработчики могли создавать пользовательские интерфейсы без особых трудностей. Почти через год после выпуска первого телефона с Android при проектировании нужно было учитывать всего лишь один размер экрана и одно значение плотности пикселов для него.

Период с конца 2009-го по начало 2010 года ознаменовался бурным ростом количества устройств, на которых работает Android. А вместе с возросшим разнообразием телефонов увеличилось и разнообразие размеров экранов и плотности пикселов.

При разработке пользовательского интерфейса важно помнить, что ваши приложения будут работать на широком спектре устройств с разными типами экранов (включая HVGA, QVGA и две разновидности WVGA: 800 × 480 и 854 × 480). Точно так же физические размеры экрана начинаются с 3,2″, включая 3,7-дюймовые Nexus One и Motorola Droid и 4-дюймовый Sony Ericsson Xperia X10.

Теперь, когда Android стал популярным, вы должны создавать свои приложения с оглядкой на еще большее количество всевозможных устройств, в перспективе это могут быть планшетные компьютеры, нетбуки и бытовая электроника.

Следующие разделы начинаются с описания типов экранов, которые нужно учитывать. Вы узнаете, как обеспечить их поддержку, и получите рекомендации по созданию приложений, которые не зависят от разрешения и плотности пикселов. В завершение научитесь тестировать свои программы на различных экранах, не имея физического доступа к телефонам.

Фреймворк для управления ресурсами и интерфейсы, не зависящие от разрешения экрана

Android предоставляет методы, с помощью которых вы можете оптимизировать пользовательский интерфейс для экранов с различными размерами и плотностью пикселов.

В этом разделе описываются спецификаторы для каталогов с ресурсами, которые можно использовать, чтобы хранить разные значения и экземпляры разметки для разных экранных конфигураций, а также элементы манифеста — с их помощью можно ограничить размеры экранов, поддерживаемые вашим приложением.

Спецификаторы ресурсов для указания размеров экрана и плотности пикселов

В главе 3 вы познакомились с фреймворком для управления ресурсами в Android. Используя этот фреймворк, можно создавать одновременно несколько структур каталогов, чтобы хранить внешние ресурсы для различных аппаратных конфигураций.

В этом разделе кратко описываются спецификаторы для имен тех каталогов, которые можно использовать при добавлении альтернативных ресурсов, чтобы поддерживать разные характеристики экрана, такие как размер, плотность пикселов и соотношение сторон.

- **Размер экрана.** Размер экрана относительно «стандартного» смартфона (например, G1 или Droid):
  - small — экран меньший, чем стандартные 3,2″;
  - medium — типичный размер экрана смартфона;
  - large — экран значительно больший, чем у типичного смартфона (например, экран планшетного компьютера или нетбука).
- **Плотность пикселов.** Описывает плотность пикселов на экране. Как правило, измеряется в точках на квадратный дюйм (dpi) и зависит от размеров экрана и его разрешения:
  - ldpi — предназначен для хранения ресурсов, рассчитанных на экраны с низкой плотностью пикселов (100–140 dpi);
  - mdpi — для экранов со средней плотностью пикселов (140–180 dpi);
  - hdpi — для экранов с высокой плотностью пикселов (190–250 dpi);
  - nodpi — для ресурсов, которые не должны масштабироваться, в зависимости от плотности пикселов на экране устройства.
- **Соотношение сторон.** Описывает отношение высоты экрана к его ширине:
  - long — для экранов, которые в альбомном режиме значительно шире, чем на стандартных смартфонах (таких как G1);
  - notlong — для экранов с обычным соотношением сторон.

Каждый из этих спецификаторов может быть использован в комбинации с любым другим или независимо от остальных, как показано в листинге 4.24.

Обратите внимание, что данные спецификаторы могут применяться в сочетании с теми, что описывались в главе 3.

Листинг 4.24. Спецификаторы для каталогов, основанные на свойствах экрана

```
res/layout-small-long/ // Разметка для маленьких, длинных экранов.
res/layout-large/      // Разметка для больших экранов.
res/drawable-hdpi/     // Ресурсы Drawable для экранов с высокой
                          плотностью пикселов.
```

Определение поддерживаемых экранных размеров

Пользовательский интерфейс некоторых приложений просто не получится оптимизировать для всех возможных типов экранов. Вы можете использовать тег `<supports-screens>` в манифесте, чтобы указать, на устройствах с какими экранами может работать ваша программа. Данный подход демонстрируется в листинге 4.25.

Листинг 4.25. Элемент манифеста, описывающий поддержку нормальных и больших экранов

```
<supports-screens
  android:smallScreens="false"
  android:normalScreens="true"
  android:largeScreens="true"
  android:anyDensity="true"
/>
```

В данном случае маленьким экраном можно назвать любой дисплей с разрешением меньше, чем HVGA. Под большим экраном подразумевается такой, который значительно больше, чем у смартфона (например, у планшетных компьютеров). Экран нормальных размеров имеет большинство смартфонов.

Атрибут `anyDensity` говорит о том, каким образом ваше приложение будет масштабироваться при отображении на устройствах с разной плотностью пикселов. Если вы учитываете это свойство экрана в своем интерфейсе (а делать это необходимо), установите этому атрибуту значение true.

При значении false Android будет использовать режим совместимости, пытаясь корректно масштабировать пользовательский интерфейс приложения. Как правило, это снижает качество изображения и приводит к артефактам при масштабировании.

Для приложений, собранных с помощью SDK с API level 4 и выше, этот атрибут по умолчанию имеет значение true.

Рекомендации по разработке интерфейсов, не зависящих от разрешения

Помимо захватывающих возможностей, предоставляемых широким разнообразием устройств под управлением Android, разработчик приложений не застрахован и от потенциальных опасностей.

В этом разделе собраны некоторые из наиболее популярных методик по созданию приложений, работающих одинаково эффективно на устройствах с любым экраном.

Главное, что вам нужно помнить: никогда не пытайтесь угадать, какой экран будет у устройства, на котором должно работать ваше приложение. Создавайте ресурсы и разметку для разных типов экранов (маленький, нормальный и большой размер в сочетании с низкой, средней и высокой плотностью пикселов), а не для конкретных разрешений или показателей dpi. Изначально ориентируясь на то, что приложение будет слегка масштабироваться на каждом устройстве, вы можете быть уверены, что пользовательский интерфейс не пострадает.

ПРИМЕЧАНИЕ

Сайт для разработчиков под Android содержит советы по поддержке разных типов экранов. Раздел Strategies for Legacy Apps чрезвычайно полезен специалистам, у которых уже есть написанные приложения и которые рассматривают возможность поддержки экранов с новыми размерами и разрешениями. Вы можете найти информацию по адресу http://developer.android.com/guide/practices/screens_support.html#strategies.

Менеджер компоновки RelativeLayout и аппаратно-независимые пикселы

По возможности следует избегать использования пиксельных значений, прописанных в коде программы. Это касается разметки, объектов Drawable и размеров шрифтов.

В частности, вы должны как можно меньше (насколько это возможно) использовать класс AbsoluteLayout, так как он предусматривает указывание пиксельных координат для каждого дочернего Представления. Вместо этого применяйте другие менеджеры компоновки, размещайте дочерние элементы друг относительно друга или основываясь на границах экрана. Для большинства сложных пользовательских интерфейсов лучше всего подойдет разметка RelativeLayout.

Внутри экземпляров своей разметки вы также должны избегать использования пиксельных значений при указании размеров для Представлений, объектов Drawable и шрифтов. Вместо этого описывайте высоту и ширину элементов, применяя атрибуты `wrap_content` или `fill_parent` везде, где это уместно. При необходимости пользуйтесь аппаратно-независимыми пикселами (density-independent pixels или dp), чтобы указывать размеры Представлений, и значениями sp (scale-independent pixels) в случае со шрифтами.

ПРИМЕЧАНИЕ

При использовании значений dp и sp размеры изображения будут корректироваться и выглядеть одинаково на устройствах, имеющих экраны с разной плотностью. Один аппаратно-независимый пиксел (dp) соответствует одному обычному пикселу на экране с плотностью 160 dpi. Линия с толщиной 2 dp при плотности 240 dpi растянется на 3 пиксела.

Использование возможностей масштабируемой графики

Ранее в этой главе вы познакомились с ресурсами Drawable, большинство из которых могут быть описаны в формате XML. Все они поддерживают плавное масштабирование во время выполнения программы независимо от размера и плотности пикселов экрана.

По возможности вместо фиксированных растровых изображений используйте следующие ресурсы Drawable:

- NinePatch;
- ShapeDrawable;
- GradientDrawable;
- композитные ресурсы и ресурсы для описания преобразований:
  - RotateDrawable и ScaleDrawable;
  - LevelListDrawable;
  - StateListDrawable.

Помните, что при описании этих ресурсов необходимо использовать аппаратно-независимые пикселы (dp).

Преимущество от использования масштабируемых ресурсов — универсальная поддержка экранов с разными размерами и разрешениями, фреймворк автоматически масштабирует ваши элементы, обеспечивая самое высокое качество изображения.

Описание ресурсов, оптимизированных для разных экранов

При использовании ресурсов Drawable, которые не могут достаточно хорошо масштабироваться динамически, вам необходимо создавать наборы изображений, оптимизированные для каждой категории пиксельной плотности (низкой, средней и высокой). Отличный пример ресурса, который должен быть оптимизирован для экранов с разной плотностью пикселов, — значки приложений.

Используя фреймворк для управления ресурсами, описанный ранее в этой главе (а также в главе 3), вы можете создавать директории со спецификаторами, чтобы хранить наборы ресурсов для всех экранов с поддерживаемой плотностью пикселов, например:

- `res/drawable-ldpi`;
- `res/drawable-mdpi`;
- `res/drawable-hdpi`.

Создавая ресурсы, оптимизированные под параметры конкретной платформы, можете быть уверены, что ваш пользовательский интерфейс останется четким и ясным, без неровностей и потерянных пикселов — типичных эффектов от масштабирования.

Стоит подумать также и о создании альтернативных ресурсов с разметкой для экранов с разными размерами. Разметка, оптимизированная для экрана обычного смартфона, может не вместить всю важную информацию, если ее использовать на маленьком телефоне, если же ее отобразить на большом устройстве, таком как планшетный компьютер, она может выглядеть слишком разреженной.

С помощью системы для управления ресурсами добавляйте спецификаторы к каталогам, создавая разметку, оптимизированную под маленькие, обычные и большие экраны:

- `res/layout-small`;
- `res/layout-normal`;
- `res/layout-large`.

Тестирование, тестирование, тестирование

На сегодняшний день представлено немало аппаратов на базе Android, имеющих экраны с разными размерами и плотностью пикселов, поэтому довольно сложно (а иногда просто невозможно) физически протестировать свое приложение на каждом устройстве.

Виртуальные устройства под управлением Android (Android Virtual Devices или AVD) — идеальная платформа для тестирования приложений с учетом разных экранных конфигураций. Виртуальные устройства также позволяют подбирать альтернативные версии платформы (1.6, 2.0, 2.1 и т. д.) и аппаратные возможности (такие как клавиатуры или трекболы).

В главе 2 вы уже научились создавать и использовать AVD, поэтому в данном разделе мы сосредоточимся на том, как лучше всего создавать виртуальные устройства с разными экранами.

Оболочки для эмулятора

Самый простой метод тестирования пользовательского интерфейса вашего приложения — использование встроенных оболочек. Каждая оболочка

эмулирует известную аппаратную конфигурацию, включая разрешение, плотность пикселов и физические размеры экрана.

В Android 2.1 для тестирования доступны следующие встроенные оболочки:

- QVGA 320 × 240, 120 dpi, 3,3″;
- WQVGA432 432 × 240, 120 dpi, 3,9″;
- HVGA 480 × 320, 160 dpi, 3,6″;
- WVGA800 800 × 480, 240 dpi, 3,9″;
- WVGA854 854 × 480, 240 dpi, 4,1″.

Тестирование экранов с нестандартными разрешениями и размерами

Одно из преимуществ AVD при тестировании устройств — возможность задавать произвольные значения для разрешения и плотности пикселов экрана.

На рис. 4.5 показан новый экземпляр AVD для устройства с разрешением 1024 × 768 и плотностью пикселов 240 dpi.

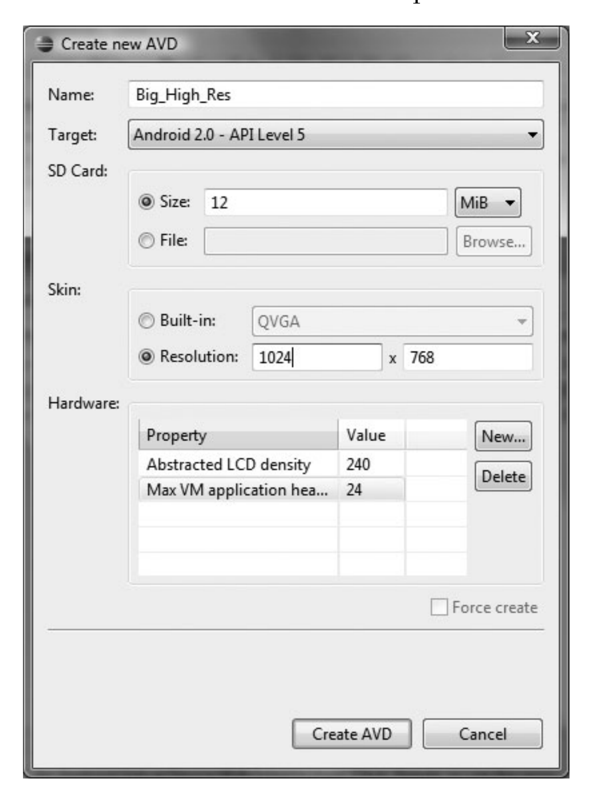


Рис. 4.5.

При запуске нового виртуального устройства появится диалоговое окно с настройками, как показано на рис. 4.6. Если вы выберете пункт Scale display to real size, указав размер экрана для нового экземпляра AVD и dpi для своего монитора, эмулятор откорректирует заданные физические размеры и плотность пикселов.

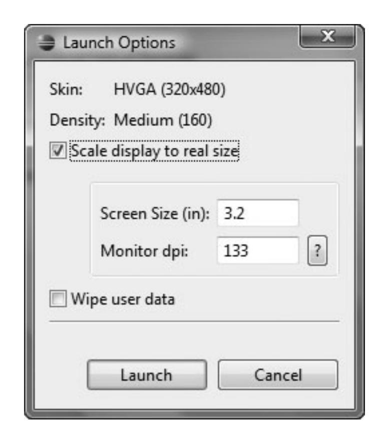


Рис. 4.6.

Благодаря этому можно оценивать пользовательский интерфейс, учитывая различные оболочки и экраны с разными размерами, разрешениями и плотностью пикселов. Это идеальный способ узнать, как ваше приложение будет выглядеть на маленьком телефоне с высоким разрешением (или на большом планшетном компьютере с разрешением поменьше).

Создание и использование меню

С помощью меню есть возможность предоставлять доступ к функциям своего приложения, не жертвуя при этом ценным экранным пространством. Каждая Активность может применять собственное меню, отображаемое при нажатии соответствующей аппаратной клавиши.

Android также поддерживает контекстные меню, которые могут быть назначены для любого Представления. Контекстное меню, как правило, появляется, когда пользователь удерживает центральную клавишу манипулятора D-pad, нажимает трекбол или длительно касается сенсорного экрана (около трех секунд). При этом Представление должно находиться в фокусе.

Главные и контекстные меню поддерживают элементы флажки (CheckBox) и переключатели (RadioButton), сокращенные клавиатурные команды, значки и дочерние меню.

Знакомство с системой управления меню в Android

Если вы когда-нибудь пытались пользоваться системой меню в смартфонах с помощью стилуса или трекбола, то знаете, что для мобильных устройств традиционный подход неудобен.

Чтобы улучшить практичность этого элемента интерфейса, Android предлагает трехступенчатую систему меню, оптимизированную для небольших экранов.

Меню со значками. Это компактное меню (рис. 4.7) появляется в нижней части экрана при нажатии кнопки Меню. Оно отображает значки и текст для ограниченного числа пунктов (как правило, не больше шести). В качестве значков рекомендуется использовать черно-белые изображения с эффектом рельефности, хотя на разных устройствах могут быть свои требования к оформлению.

Это меню не может содержать флажки и переключатели, также в нем нет поддержки сочетаний клавиш для отдельных пунктов. В связи с этим лучше не использовать элементы CheckBox и RadioButton внутри объектов MenuItem, потому как они все равно не будут отображаться.

Когда количество пунктов в меню Активности превышает допустимое значение, появляется объект MenuItem с надписью More. Если его выбрать, на экране отобразится расширенное меню. При нажатии клавиши возврата меню со значками закрывается.

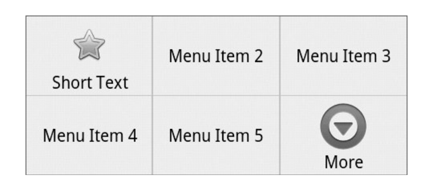


Рис. 4.7.

Расширенное меню. Выводится на экран, когда пользователь выбирает пункт More из меню со значками. Расширенное меню (рис. 4.8) отображает прокручиваемый список, содержащий только те элементы, которые не были показаны в меню со значками. При этом на экран выводится полный текст, сокращенные клавиатурные команды и элементы CheckBox/RadioButton.

Значки здесь не отображаются. Нажав клавишу возврата, пользователь вернется в меню со значками.

ВНИМАНИЕ

Вы не способны заставить Android отображать расширенное меню вместо обычного. Поэтому нужно уделить особое внимание пунктам, содержащим элементы CheckBox и RadioButton. Максимальное количество

значков может быть разным на различных устройствах, поэтому рекомендуется уведомлять об их статусе с помощью пиктограмм или изменений в тексте.

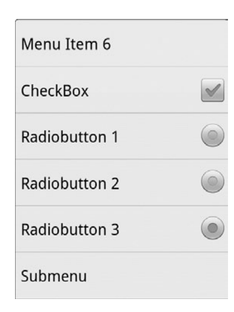


Рис. 4.8.

Дочерние меню. Традиционные иерархические меню могут быть трудными в использовании, даже если работать с ними мышью, поэтому неудивительно, что данная абстракция абсолютно непригодна на мобильных устройствах. В качестве альтернативы Android отображает каждое дочернее меню во всплывающем окне.

К примеру, когда пользователь выбирает дочернее меню (как то, которое показано на рис. 4.8), его элементы отображаются во всплывающем диалоговом окне, как на рис. 4.9.

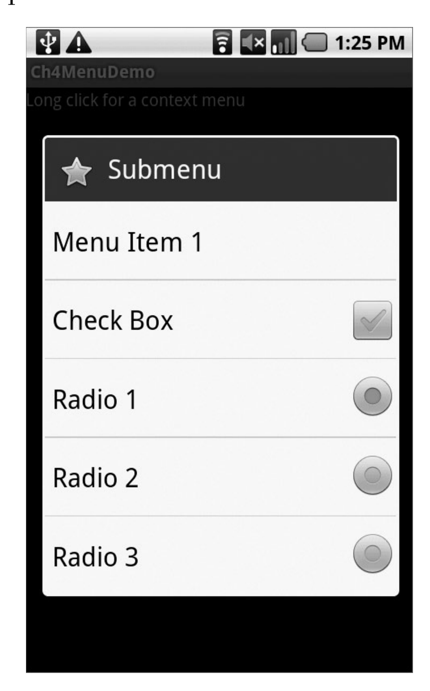


Рис. 4.9.

Обратите внимание, что название дочернего меню выводится в заголовочной строке, а в каждом пункте показаны флажки и переключатели (если таковые имеются), сокращенные клавиатурные команды и полный текст. Поскольку Android в данном случае не поддерживает вложенность, вы не можете добавить одно дочернее меню в другое (это приведет к выбросу исключения).

Как и в предыдущем случае, значки в дочернем меню не отображаются, поэтому для пунктов данного типа меню их рекомендуется не указывать.

Если нажать кнопку Назад, всплывающее окно закроется без перехода к расширенному меню или меню со значками.

Описание меню для Активности

Чтобы описать меню для Активности, переопределите обработчик onCreateOptionsMenu. Этот метод инициирует первое появление меню на экране.

Обработчик onCreateOptionsMenu принимает в качестве параметра объект Menu. Вы можете сохранить ссылку на меню и использовать ее в любом месте кода, пока метод onCreateOptionsMenu опять не будет вызван.

Вам необходимо всегда использовать реализацию этого обработчика из родительского класса, потому как она при необходимости автоматически включает в меню дополнительные системные пункты.

Используйте метод add из объекта Menu, чтобы заполнить меню. Для каждого нового пункта вы должны указывать параметры.

- Группу, чтобы выделить пункты для пакетной обработки и сортировки.
- Уникальный идентификатор для каждого пункта. Выбор элементов, как правило, обрабатывается с помощью метода onOptionsItemSelected, поэтому данный параметр важен при определении, какой именно пункт был указан. Каждый ID для меню принято описывать в виде приватного статического свойства внутри класса Активности. Вы можете использовать статическую константу Menu.FIRST и просто увеличивать значение для каждого следующего элемента.
- Значение, описывающее порядок, в котором пункты меню будут выведены на экран.
- Текст для пункта меню, указывается либо в виде строки, либо с помощью строкового ресурса.

Закончив наполнять меню, верните значение true.

В листинге 4.26 показано добавление в меню Активности единственного пункта.

Листинг 4.26. Добавление пункта меню

```
static final private int MENU_ITEM = Menu.FIRST;

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  // Идентификатор группы
  int groupId = 0;
  // Уникальный идентификатор для пункта меню. Используется
  // при обработке событий.
  int menuItemId = MENU_ITEM;
  // Порядок размещения пункта в меню
  int menuItemOrder = Menu.NONE;
  // Текст, который будет выведен для этого пункта меню.
  int menuItemText = R.string.menu_item;

  // Создайте пункт меню и сохраните ссылку на него.
  MenuItem menuItem = menu.add(groupId, menuItemId,
                               menuItemOrder, menuItemText);

  return true;
}
```

Как и в случае с самим меню, каждый объект MenuItem, возвращаемый методом add, действителен до следующего вызова обработчика onCreateOptionsMenu. Вместо того чтобы сохранять ссылку на каждый пункт, можно найти конкретный объект MenuItem, передавая его идентификатор в метод findItem из экземпляра меню.

Параметры пунктов меню

Android поддерживает большинство традиционных параметров для пунктов меню, с которыми вы, скорее всего, уже знакомы. Это касается значков, ярлыков, флажков и переключателей.

Элементы CheckBox и RadioButton. Эти элементы видны в расширенном и дочернем меню (см. рис. 4.9). Чтобы пункт меню отображался в виде флажка, используйте метод setCheckable. Состояние этих элементов контролируется с помощью метода setChecked.

Группа переключателей — это набор пунктов, из которых в каждый момент времени может быть выбран только один. При указании одного такого пункта автоматически сбрасывается выбор для любого другого в той же группе.

Чтобы создать группу для элементов RadioButton, назначьте один и тот же групповой идентификатор для каждого пункта, передав его в метод Menu.setGroupCheckable и установив для параметра exclusive значение true.

Элементы CheckBox не отображаются в меню со значками, поэтому объекты MenuItem такого вида должны быть зарезервированы для дочер-

него и расширенного меню. В следующем фрагменте кода показан процесс добавления флажков и группы переключателей:

```
// Создайте новый пункт с элементом CheckBox.
menu.add(0, CHECKBOX_ITEM, Menu.NONE, "CheckBox").setCheckable(true);

// Создайте новую группу для элементов RadioButton.
menu.add(RB_GROUP, RADIOBUTTON_1, Menu.NONE, "Radiobutton 1");
menu.add(RB_GROUP, RADIOBUTTON_2, Menu.NONE, "Radiobutton 2");
menu.add(RB_GROUP, RADIOBUTTON_3, Menu.NONE,
   "Radiobutton 3").setChecked(true);
menu.setGroupCheckable(RB_GROUP, true, true);
```

Сокращенные клавиатурные команды. Вы можете задать клавиатурные ярлыки для объектов MenuItem, используя метод setShortcut, который принимает два сочетания клавиш: одно для цифровой клавиатуры, другое — для полной. Клавиши нечувствительны к регистру.

```
// Добавьте ярлык для этого пункта меню. "0" - для цифровой клавиатуры,
// "b" - для полной.
menuItem.setShortcut('0', 'b');
```

Краткие заголовки. Меню со значками не отображает ярлыки или переключатели, поэтому часто необходимо изменять выводимый им текст, чтобы сигнализировать о его состоянии. Метод setTitleCondensed позволяет указать текст, который будет отображаться только в меню со значками.

```
menuItem.setTitleCondensed("Short Title");
```

Значки. Свойство icon — это идентификатор для ресурса Drawable, содержащий значок, который применяется в объекте MenuItem. Значки отображаются только в соответствующем меню, в расширенном и дочернем меню они не выводятся. Вы можете указать любой ресурс Drawable в качестве значка, хотя для меню, как правило, рекомендуется использовать черно-белые изображения с эффектом рельефности.

```
menuItem.setIcon(R.drawable.menu_item_icon);
```

OnMenuItemClickListener. Обработчик, срабатывающий при выборе пункта меню. Из соображений эффективности использовать его не рекомендуется. Вместо этого за выбор пунктов меню должен отвечать обработчик onOptionsItemSelected, как показано далее в этом разделе.

```
menuItem.setOnMenuItemClickListener(new OnMenuItemClickListener() {
  public boolean onMenuItemClick(MenuItem _menuItem) {
    [ ... выполните обработку нажатия, верните true в случае успеха ... ]
    return true;
  }
});
```

Намерения. Намерение, назначенное для пункта меню, срабатывает при нажатии этого пункта, если данное событие не было перехвачено обработчиками

onMenuItemClickListener или onOptionsItemSelected. Сработав, Намерение передается в метод startActivity.

```
menuItem.setIntent(new Intent(this, MyOtherActivity.class));
```

Динамическое изменение пунктов меню

Переопределяя метод onPrepareOptionsMenu из своей Активности, вы можете менять меню в зависимости от текущего состояния приложения, прежде чем оно будет выведено на экран. Это позволяет вам динамически делать пункты доступными/недоступными, видимыми/невидимыми, изменять их текст.

Динамически изменять объекты MenuItem можно двумя способами: находить ссылку на пункты меню в методе onCreateOptionsMenu при их создании либо использовать метод findItem из объекта Menu. В листинге 4.27 показан второй вариант, предусматривающий переопределение обработчика onPrepareOptionsMenu.

Листинг 4.27. Динамическое изменение меню

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
  super.onPrepareOptionsMenu(menu);

  MenuItem menuItem = menu.findItem(MENU_ITEM);

  [ ... изменение пунктов меню ... ]

  return true;
}
```

Выбор пунктов меню

Android контролирует все события, связанные с выбором пунктов в меню Активностей, используя единственный обработчик onOptionsItemSelected. Назначенный пункт передается этому методу в виде параметра MenuItem.

Чтобы реагировать на выбор конкретных пунктов, необходимо сравнивать значение item.getItemId с идентификаторами объектов MenuItem, которые вы использовали при заполнении меню, и выполнять соответствующий код, как показано в листинге 4.28.

Листинг 4.28. Обработка событий, отвечающих за выбор пунктов меню

```
public boolean onOptionsItemSelected(MenuItem item) {
  super.onOptionsItemSelected(item);

  // Определите, какой именно пункт меню был выбран
```

```
    switch (item.getItemId()) {

      // Сделайте перебор по всем известным пунктам
      case (MENU_ITEM):
        [ ... Выполните обработку ... ]
        return true;
    }

    // Верните false, если вы не обработали это событие.
    return false;
  }
```

Дочерние и контекстные меню

Контекстные меню используют те же всплывающие окна, что и дочерние (см. рис. 4.9). И хотя внешне они не отличаются, заполняются разными способами.

Создание дочерних меню

Дочерние меню отображаются в виде обычных объектов MenuItems, при выборе которых появляются новые пункты. В традиционных системах дочерние меню выводятся на экран в качестве древовидной иерархии. Но Android использует другой подход, упрощая процесс навигации по меню на устройствах с небольшими экранами. Вместо древовидной структуры при выборе дочернего меню появляется единственное всплывающее окно, отображающее все пункты.

Добавить дочернее меню можно с помощью метода addSubMenu. Он поддерживает тот же набор параметров, что и метод add, применяемый при добавлении обычных пунктов, позволяет указывать группу, уникальный идентификатор и текстовую строку для каждого дочернего меню. Вы также можете использовать методы setHeaderIcon и setIcon для задания изображений, которые будут выводиться в заголовочной строке всплывающего окна и в меню со значками соответственно.

Объекты MenuItem в данном случае поддерживают те же параметры, что и пункты для других видов меню (расширенных и со значками). Однако в отличие от традиционных систем вложенные меню в Android не поддерживаются.

В следующем фрагменте кода представлена выдержка из реализации обработчика onCreateMenuOptions — добавление дочернего меню в главное. При этом для дочернего меню создается новый пункт и устанавливается заголовочный значок.

```
SubMenu sub = menu.addSubMenu(0, 0, Menu.NONE, "Submenu");
sub.setHeaderIcon(R.drawable.icon);
sub.setIcon(R.drawable.icon);

MenuItem submenuItem = sub.add(0, 0, Menu.NONE, "Submenu Item");
```

Использование контекстного меню

Контекстное меню привязывается к выделенному Представлению и выводится на экран при нажатии трекбола, средней кнопки манипулятора D-pad или на элемент интерфейса (последний должен удерживаться в течение примерно трех секунд).

Заполнение контекстных и главных меню во многом похоже. Есть два способа создать контекстное меню для конкретного Представления.

Создание контекстного меню. Первый подход — создание обобщенного объекта ContextMenu для класса View, в котором переопределяется обработчик onCreateContextMenu, как показано ниже:

```
@Override
public void onCreateContextMenu(ContextMenu menu) {
  super.onCreateContextMenu(menu);
  menu.add("ContextMenuItem1");
}
```

Контекстное меню будет доступно внутри Активности, содержащей этот класс View.

Шире используется создание контекстных меню для конкретных Активностей путем переопределения обработчика onCreateContextMenu и регистрации с помощью метода registerForContextMenu Представлений, которые это меню могут использовать. Данный процесс показан в листинге 4.29.

Листинг 4.29. Назначение контекстного меню для Представления

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  EditText view = new EditText(this);
  setContentView(view);

  registerForContextMenu(view);
}
```

После регистрации Представления обработчик onCreateContextMenu будет срабатывать при первом отображении контекстного меню для этого объекта View.

Переопределите метод onCreateContextMenu и проверьте, какое именно Представление его вызвало, чтобы иметь возможность заполнить меню соответствующими пунктами, как показано в дополнении к листингу 4.29:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
```

```
    super.onCreateContextMenu(menu, v, menuInfo);

    menu.setHeaderTitle("Context Menu");
    menu.add(0, menu.FIRST, Menu.NONE,
             "Item 1").setIcon(R.drawable.menu_item);
    menu.add(0, menu.FIRST+1, Menu.NONE, "Item 2").setCheckable(true);
    menu.add(0, menu.FIRST+2, Menu.NONE, "Item 3").setShortcut('3', '3');
    SubMenu sub = menu.addSubMenu("Submenu");
    sub.add("Submenu Item");
  }
```

Как видно из примера классы ContextMenu и Menu поддерживают один и тот же метод add, поэтому способ заполнения этих видов меню ничем не отличается. Заметьте, что значки при этом отображаться не будут. Однако вы можете указать название и значок, которые будут выводиться в заголовочной строке контекстного меню.

Android также поддерживает динамическое заполнение контекстных меню с помощью Фильтров намерений. Данный механизм позволяет заполнять контекстные меню, указывая, какой тип данных представляет текущий объект View, и опрашивая другие приложения на предмет каких-либо действий для него.

Наиболее яркий пример этого механизма — пункты меню вырезать/копировать/вставить, доступные в элементах EditText. Использование Фильтров намерений для заполнения контекстных меню подробно рассматривается в следующей главе.

Выбор пунктов в контекстном меню обрабатывается так же, как и в меню для Активностей. Вы можете прикреплять Намерения или интерфейс OnMenuItemClickListener напрямую к каждому объекту MenuItem, а также воплотить более предпочтительный подход — переопределение метода onContextItemSelected, принадлежащего Активности.

Данный обработчик событий срабатывает каждый раз, когда в контекстном меню выбирается какой-то пункт.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  super.onContextItemSelected(item);

  [ ... Обработка выбора пункта в меню ... ]

  return false;
}
```

Описание меню с помощью XML

Android позволяет описывать иерархические меню в виде ресурсов в формате XML.

Как и в случае с разметкой и другими ресурсами, это дает возможность создавать разные меню для разных аппаратных конфигураций, языковых настроек или регионов. К примеру, вы хотите переместить некоторые экранные настройки в меню, но только для устройств с маленьким дисплеем.

Меню описываются в формате XML и хранятся в каталоге res/menu внутри вашей директории с ресурсами. Все иерархии должны создаваться в виде отдельных файлов, имена которых (в нижнем регистре) становятся идентификаторами ресурсов.

Создайте иерархическое меню, используя тег <menu> в качестве главного узла, а также набор тегов <item> для описания каждого пункта. Все узлы item поддерживают атрибуты, с помощью которых можно задать свойства для объектов MenuItem, включая текст, значок, сокращенную клавиатурную команду и настройки для переключателя.

Чтобы создать дочернее меню, добавьте новый тег <menu> внутрь узла <item>.

В листинге 4.30 показано, как с помощью XML-ресурсов создать иерархическое меню, описанное в листинге 4.29.

Листинг 4.30. Описание меню в формате XML

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  android:name="Context Menu">
  <item
    android:id="@+id/item01"
    android:icon="@drawable/menu_item"
    android:title="Item 1">
  </item>
  <item
    android:id="@+id/item02"
    android:checkable="true"
    android:title="Item 2">
    </item>
  <item
    android:id="@+id/item03"
    android:numericShortcut="3"
    android:alphabeticShortcut="3"
    android:title="Item 3">
  </item>
  <item
    android:id="@+id/item04"
    android:title="Submenu">
    <menu>
      <item
        android:id="@+id/item05"
        android:title="Submenu Item">
      </item>
    </menu>
  </item>
</menu>
```

Воспользоваться данным ресурсом можно с помощью класса MenuInflator, применяя его внутри обработчиков onCreateOptionsMenu или onCreateContextMenu, как показано в листинге 4.31.

Листинг 4.31. Получение меню из XML-ресурса

```
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.my_menu, menu);
  menu.setHeaderTitle("Context Menu");
}
```

Совершенствование приложения To-Do List

В следующем примере[1] вы добавите поддержку простого меню в приложение To-Do List, которое создали в главе 2 (и улучшили в предыдущих разделах текущей главы).

Вы добавите возможность удалять элементы из списка задач с помощью контекстного и главного меню, улучшите использование экранного пространства, отображая строку ввода только при добавлении нового пункта.

1. Начните с класса Активности ToDoList. Импортируйте пакеты, которые вам понадобятся для поддержки функциональности, связанной с меню.

```
import android.view.Menu;
import android.view.MenuItem;
import android.view.ContextMenu;
import android.widget.AdapterView;
```

2. Добавьте приватные статические финальные свойства, содержащие уникальные идентификаторы для каждого пункта меню.

```
static final private int ADD_NEW_TODO = Menu.FIRST;
static final private int REMOVE_TODO = Menu.FIRST + 1;
```

3. Переопределите метод onCreateOptionsMenu, создав два новых пункта меню для добавления и удаления элементов в списке задач. Укажите соответствующий текст, назначьте ресурсы со значками и сокращенные клавиатурные команды для каждого пункта.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  // Создайте новые пункты меню.
```

[1] Все фрагменты кода в этом примере — часть проекта Todo List 2 из главы 4, их можно загрузить на сайте Wrox.com.

```
    MenuItem itemAdd = menu.add(0, ADD_NEW_TODO, Menu.NONE,
                                R.string.add_new);
    MenuItem itemRem = menu.add(0, REMOVE_TODO, Menu.NONE,
                                R.string.remove);

    // Установите значки
    itemAdd.setIcon(R.drawable.add_new_item);
    itemRem.setIcon(R.drawable.remove_item);

    // Назначьте сокращенные клавиатурные команды для каждого пункта.
    itemAdd.setShortcut('0', 'a');
    itemRem.setShortcut('1', 'r');

    return true;
  }
```

Если запустить Активность и нажать аппаратную кнопку Меню, на экране должно отобразиться меню, как показано на рис. 4.10.

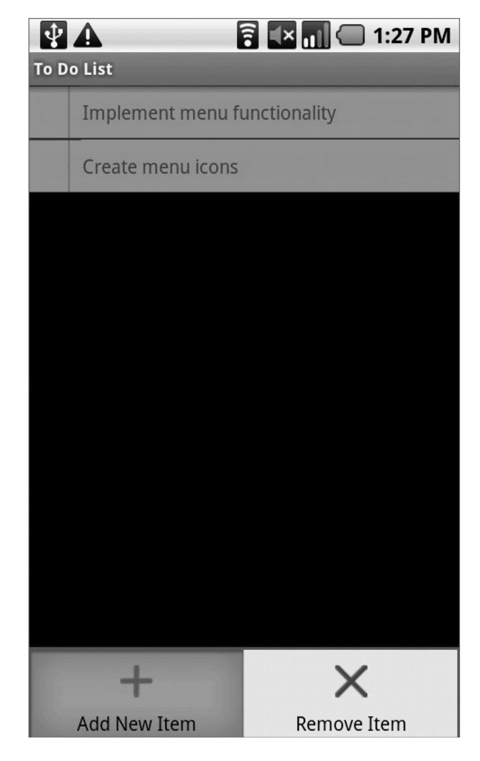


Рис. 4.10.

4. После заполнения меню для Активности нужно создать еще одно меню — контекстное. Сперва измените метод onCreate, зарегистрировав элемент ListView, который будет использоваться в контекстном меню. Затем переопределите обработчик onCreateContextMenu, добавив в объект ContextMenu пункт удалить.

```
@Override
public void onCreate(Bundle savedInstanceState) {


  [ ... ранее написанный код для метода onCreate ... ]

  registerForContextMenu(myListView);
}

@Override
public void onCreateContextMenu(ContextMenu menu,
                                View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);

  menu.setHeaderTitle("Selected To Do Item");
  menu.add(0, REMOVE_TODO, Menu.NONE, R.string.remove);
}
```

5. Теперь нужно сделать так, чтобы внешний вид меню изменялся в зависимости от программного контекста. Для этого переопределите метод onPrepareOptionsMenu. Объект MenuItem должен быть изменен, чтобы во время добавления новой задачи показывать надпись Cancel вместо Delete.

```
private boolean addingNew = false;

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
  super.onPrepareOptionsMenu(menu);

  int idx = myListView.getSelectedItemPosition();

  String removeTitle = getString(addingNew ?
                                 R.string.cancel : R.string.remove);

  MenuItem removeItem = menu.findItem(REMOVE_TODO);
  removeItem.setTitle(removeTitle);
  removeItem.setVisible(addingNew || idx > -1);

  return true;
}
```

6. Чтобы код из предыдущего пункта работал, необходимо, чтобы элементы todoListItems и ListView были видны за пределами метода onCreate. То же самое сделайте и для ArrayAdapter и EditText для добавления поддержки действий добавить и удалить, которые будут реализованы позже.

```
private ArrayList<String> todoItems;
private ListView myListView;
private EditText myEditText;
private ArrayAdapter<String> aa;

@Override
public void onCreate(Bundle savedInstanceState) {
```

```
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  // Получите ссылки на элементы пользовательского интерфейса
     myListView = (ListView)findViewById(R.id.myListView);
  myEditText = (EditText)findViewById(R.id.myEditText);

  todoItems = new ArrayList<String>();
  int resID = R.layout.todolist_item;
  aa = new ArrayAdapter<String>(this, resID, todoItems);
  myListView.setAdapter(aa);

  myEditText.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
      if (event.getAction() == KeyEvent.ACTION_DOWN)
        if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER)
        {
          todoItems.add(0, myEditText.getText().toString());
          myEditText.setText("");
          aa.notifyDataSetChanged();
          return true;
        }
      return false;
    }
  });

  registerForContextMenu(myListView);
}
```

7. Необходимо контролировать выбор пунктов в меню. Переопределите методы onOptionsItemSelected и onContextItemSelected, чтобы вызывать заглушки, которые обрабатывают новые объекты MenuItem.

   7.1. Начните с переопределения обработчика onOptionsItemSelected, чтобы реагировать на выделение пунктов в главном меню. Для пункта, который отвечает за удаление, можете использовать метод getSelectedItemPosition из объекта ListView, чтобы находить выбранный элемент.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  super.onOptionsItemSelected(item);

  int index = myListView.getSelectedItemPosition();

  switch (item.getItemId()) {
    case (REMOVE_TODO): {
      if (addingNew) {
        cancelAdd();
      }
      else {
        removeItem(index);
      }
      return true;
    }
```

```
    case (ADD_NEW_TODO): {
      addNewItem();
      return true;
    }
  }

  return false;
}
```

7.2. Переопределите метод onContextItemSelected, чтобы обрабатывать выбор пунктов в контекстном меню. Имейте в виду, что применяемая реализация интерфейса ContextMenuInfo привязана к классу AdapterView. Она содержит ссылки на Представление, которое вызвало контекстное меню, и индекс для данных, принадлежащих исходному Адаптеру (Adapter). Используйте последний в качестве индекса элемента, который нужно удалить.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  super.onContextItemSelected(item);
  switch (item.getItemId()) {
    case (REMOVE_TODO): {
       AdapterView.AdapterContextMenuInfo menuInfo;
       menuInfo =(AdapterView.AdapterContextMenuInfo)item.getMenuInfo();
       int index = menuInfo.position;

      removeItem(index);
      return true;
    }
  }
  return false;
}
```

7.3. Создайте заглушки, которые вызываются в обработчике onContextItemSelected, созданном выше.

```
private void cancelAdd() {
}

private void addNewItem() {
}

private void removeItem(int _index) {
}
```

8. Реализуйте каждую заглушку, чтобы предоставить новую функциональность.

```
private void cancelAdd() {
    addingNew = false;
  myEditText.setVisibility(View.GONE);
}

private void addNewItem() {
  addingNew = true;
```

```
    myEditText.setVisibility(View.VISIBLE);
    myEditText.requestFocus();
  }

  private void removeItem(int _index) {
    todoItems.remove(_index);
    aa.notifyDataSetChanged();
  }
```

9. После того как новая задача добавлена, необходимо скрыть строку ввода. Измените обработчик onKeyListener в методе onCreate, чтобы он вызывал функцию cancelAdd после добавления нового элемента.

```
myEditText.setOnKeyListener(new OnKeyListener() {
  public boolean onKey(View v, int keyCode, KeyEvent event) {
    if (event.getAction() == KeyEvent.ACTION_DOWN)
      if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER)
      {
        todoItems.add(0, myEditText.getText().toString());
        myEditText.setText("");
        aa.notifyDataSetChanged();
        cancelAdd();
        return true;
      }
    return false;
  }
});
```

10. В завершение сделайте пользовательский интерфейс логичным и последовательным, изменив разметку в файле `main.xml` таким образом, чтобы строка ввода была скрыта, пока пользователь не решит добавить новый элемент:

```
<EditText
  android:id="@+id/myEditText"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text=""
  android:visibility="gone"
/>
```

Запустив приложение, вы получите возможность вызывать главное меню, чтобы добавлять или удалять элементы списка. Каждый элемент содержит контекстное меню, с помощью которого его можно удалить.

Резюме

Теперь вы знакомы с основными подходами к созданию интуитивно понятных пользовательских интерфейсов для приложений в Android. Вы изучили Представления и разметку, познакомились с системой для управления меню.

Научились создавать экраны с Активностями, позиционируя Представления с помощью менеджеров компоновки (можно создавать как в коде программы, так и в виде ресурсов). Вы узнали, как расширять, группировать и создавать новые элементы, основанные на классе View, чтобы иметь возможность обеспечивать нестандартный внешний вид и поведение для своих приложений.

В этой главе вы также:

- познакомились с некоторыми элементами управления и виджетами, которые являются частью Android SDK;
- узнали, как использовать свои нестандартные Представления внутри Активностей;
- изучили создание динамических ресурсов Drawable в формате XML;
- научились создавать пользовательские интерфейсы, которые не зависят от разрешения и плотности пикселов на экране;
- узнали, как создавать и использовать главные и контекстные меню;
- расширили возможности приложения To-Do List, добавив в него поддержку нестандартных Представлений и функции меню;
- создали с нуля новый элемент CompassView.

После рассмотрения основ проектирования пользовательского интерфейса в Android можно перейти к следующей главе, которая посвящена связыванию компонентов приложения с помощью Намерений, Широковещательных приемников и Адаптеров. Вы узнаете, как запускать новые Активности, передавать и принимать запросы на выполнение действий, научитесь взаимодействовать с Интернетом, а также рассмотрите класс Dialog.

Глава 5
НАМЕРЕНИЯ, ШИРОКОВЕЩАТЕЛЬНЫЕ ПРИЕМНИКИ, АДАПТЕРЫ И ИНТЕРНЕТ

Содержание главы

- Знакомство с Намерениями.
- Запуск обычных и дочерних Активностей с помощью явных и неявных Намерений.
- Фильтры намерений и процесс утверждения Намерений.
- Использование Linkify.
- Намерения, транслируемые действия и Широковещательные намерения.
- Использование Адаптеров (Adapters) для привязки данных к Представлениям.
- Использование Интернета в Android.
- Изменение и использование диалоговых окон.

На первый взгляд темы, рассматриваемые в этой главе, могут показаться довольно простыми, но на практике они играют роль связующего звена между приложениями и их компонентами.

На большинстве платформ мобильные приложения работают в собственных «песочницах». Они изолированы друг от друга и жестко ограничены во взаимодействии с аппаратными и системными компонентами. На платформе Android все аналогично, но здесь эти ограничения можно преодолеть с помощью Намерений, Приемников широковещательных намерений, Адаптеров, Источников данных и сети Интернет.

В этой главе вы узнаете, что такое Намерения. Их можно назвать наиболее уникальной и важной концепцией при разработке под Android. Вы научитесь использовать их для обмена данными между приложениями и компонентами, для запуска Активностей или Сервисов (явным и неявным образом, а также с помощью связывания во время выполнения программы).

Используя Намерения по умолчанию, научитесь инициировать выполнение какого-либо действия, позволяя системе самой выбирать наиболее подходящий Сервис.

Широковещательные намерения используются для оповещения о событиях в рамках всей системы. Вы узнаете, каким образом их можно передавать и принимать, используя Приемники широковещательных намерений.

Вы изучите Адаптеры и методы их применения, чтобы связывать уровень Представления с источниками данных. Затем перейдете к рассмотрению диалоговых окон.

После знакомства с механизмами передачи и потребления данных вашему вниманию будет представлена модель интернет-подключений, функционирующая в Android, а также некоторые методы обработки потоков данных из сети, доступные в Java.

На примере приложения для отслеживания землетрясений будет показано, как связать все вышеперечисленные возможности воедино. Данный демонстрационный проект станет основой для дальнейших улучшений и расширений, которые вы станете добавлять в него в последующих главах.

Знакомство с Намерениями

Намерения используются в качестве механизма передачи сообщений. Этот механизм может работать как внутри вашего приложения, так и на межпроцессном уровне. Намерения могут применяться:

- для объявления о запуске Активности или Сервиса, направленных на выполнение каких-либо действий (как правило, речь о работе с определенной частью данных);
- передачи уведомлений о том, что произошло некое событие (или действие);
- явного запуска определенного Сервиса или Активности.

Вы можете задействовать Намерения для поддержки взаимодействия между программными компонентами, установленными на устройстве под управлением Android, неважно, частью какого приложения они являются. Благодаря этому ваше устройство из платформы, содержащей набор независимых компонентов, превратится в единую интегрированную систему.

Наиболее частый пример использования Намерений — запуск новых Активностей явным образом (с указанием класса для загрузки) или неявным (с помощью объявления о выполнении заданного действия). Во втором случае действие выполняется за пределами Активности, которая его инициировала, и даже за пределами исходного приложения.

Намерения также могут применяться для трансляции сообщений по системе. Любое приложение способно зарегистрировать Широковещательный приемник и отслеживать эти Намерения с возможностью на них реагировать. Это позволяет создавать приложения, использующие событийную модель,

в основе которой лежат внутренние, системные или сторонние события, передаваемые внешними программами.

Android транслирует Намерения для объявления о системных событиях, например об изменениях в состоянии сетевого подключения или в уровне заряда батареи. Системные приложения в Android, такие как программы дозвона или управления SMS, регистрируют компоненты, отслеживающие заданные Намерения, например входящий звонок или получено новое текстовое сообщение, и соответствующим образом реагируют на них.

Использование Намерений для распространения действий (даже внутри одного приложения) — фундаментальный проектировочный принцип для Android, который поощряет создание слабо связанных между собой компонентов, благодаря чему упрощается замена разных частей приложения. Это также способствует утверждению простой модели для расширения возможностей приложений.

Использование Намерений для запуска Активностей

Чаще всего Намерения используются для связывания компонентов приложения. С их помощью можно запускать Активности и осуществлять переходы между ними.

ПРИМЕЧАНИЕ

Методы, приводимые в данном разделе, относятся к запуску новых Активностей, но этот же подход применим и к Сервисам. Подробности о запуске (и создании) Сервисов смотрите в главе 9.

Чтобы запустить Активность, вызовите метод startActivity, передав ему в качестве параметра Намерение, как показано в следующем фрагменте:

```
startActivity(myIntent);
```

Намерение может явно указывать на класс Активности, экземпляр которого должен быть запущен, или содержать действие, которое эта Активность должна выполнить. Во втором случае система сама динамически выберет Активность, задействовав процесс, называемый Утверждением намерения (Intent Resolution).

Метод startActivity находит и запускает одиночную Активность, которая лучше всего подходит для вашего Намерения.

При использовании этого метода ваше приложение не получит никаких уведомлений о завершении работы запущенного компонента. Если вам необходимо увидеть результат выполнения Активности, воспользуйтесь методом startActivityForResult, который более подробно описан в следующем разделе.

Запуск новых Активностей явным образом

В главе 2 вы узнали, что приложения состоят из ряда взаимосвязанных экранов — Активностей, которые необходимо включить в манифест приложения. Чтобы соединить их воедино, вы должны явным образом указывать, какую Активность хотите открыть.

Для этого требуется создать новое Намерение, указав текущий контекст приложения и класс Активности, экземпляр которой вы хотите запустить. Передайте это Намерение в метод startActivity, как показано в листинге 5.1.

Листинг 5.1. Явный запуск Активности

```
Intent intent = new Intent(MyActivity.this, MyOtherActivity.class);
startActivity(intent);
```

После вызова метода startActivity создастся новая Активность (в данном случае MyOtherActivity), она станет видимой и переместится на вершину стека, содержащего работающие компоненты.

При вызове метода finish из новой Активности (или при нажатии аппаратной клавиши возврата) она будет закрыта и удалена из стека. Разработчик также может перемещаться к предыдущей (или к любой другой) Активности, используя все тот же метод startActivity.

Неявные Намерения и связывание во время выполнения программы

Неявные Намерения — это механизм, позволяющий запрашивать анонимные компоненты приложений с помощью действий. Это значит, что вы можете попросить систему запустить Активность, выполняющую заданное действие, не зная ничего ни о самой Активности, ни о ее приложении.

Создавая новое неявное Намерение для передачи в метод startActivity, необходимо назначить действие, которое должно выполниться, а также при желании указать вспомогательный путь URI к тем данным, что нужно обработать. Вы также можете передать дополнительные данные в целевую Активность, используя параметр Намерения extras.

При применении этого Намерения для запуска Активности Android во время работы приложения сам найдет класс компонента, который лучше всего подходит для заданного действия, учитывая указанный тип данных. Это значит, что вы можете создавать проекты, используя возможности других приложений и не зная при этом, как они называются и какую функциональность предоставляют.

Например, чтобы позволить пользователям звонить из вашего приложения, можно создать собственную программу дозвона или применить неявное Намерение, указав действие (звонок), которое необходимо выполнить

(телефонный номер дается в виде пути URI). Этот процесс демонстрируется в листинге 5.2.

Листинг 5.2. Неявный запуск Активности

```
if (somethingWeird && itDontLookGood) {
  Intent intent = new Intent(Intent.ACTION_DIAL, Uri.
parse("tel:555.2368"));
  startActivity(intent);
}
```

Android получает это Намерение и запускает Активность, с помощью которой можно сделать звонок по указанному номеру — в данном случае это Активность для дозвона.

Если есть сразу несколько Активностей, предоставляющих телефонные функции, пользователю предложат выбрать одну из них. Полный процесс утверждения Намерений описан далее в этой главе.

Различные системные приложения предоставляют Активности для действий с конкретными видами данных. Сторонние приложения, включая ваши собственные, могут быть зарегистрированы для поддержки как новых, так и уже существующих системных действий. Позже в этой главе вы познакомитесь с некоторыми системными действиями и узнаете, как регистрировать собственные Активности для их поддержки.

Возвращение результатов из Активностей

Активность, запущенная с помощью метода startActivity, теряет зависимость от «родителя» и не возвращает никаких результатов при закрытии.

Вы можете запустить Активность в виде дочернего объекта, неразрывно связанного с родительским экземпляром. При закрытии дочерняя Активность инициирует срабатывание обработчика внутри родительской. Такой механизм идеально подходит в ситуациях, когда одна Активность предоставляет возможность ввода данных (например, выбор элемента списка пользователем) для другой.

Дочерние Активности отличаются от любых других способом запуска. Они должны быть зарегистрированы в манифесте приложения. Фактически любая Активность, упоминающаяся в манифесте, может стать дочерней. Это относится и к системным, и к сторонним Активностям.

Запуск дочерних Активностей. Метод startActivityForResult работает так же, как и startActivity, но с одним важным отличием. Помимо явных или неявных Намерений, используемых для определения нужной Активности, вы также можете передавать код запроса. Позже это значение применяется для однозначной идентификации дочерней Активности, которая вернула результат.

Каркас для запуска дочерней Активности показан в листинге 5.3.

Листинг 5.3. Запуск Активности, которая должна вернуть результат

```
private static final int SHOW_SUBACTIVITY = 1;

Intent intent = new Intent(this, MyOtherActivity.class);
startActivityForResult(intent, SHOW_SUBACTIVITY);
```

Дочерняя Активность, как и любая другая, может быть запущена явным и неявным образом. В листинге 5.4, чтобы запустить Активность для выбора контакта, используется неявное Намерение.

Листинг 5.4. Неявный запуск Активности, которая должна вернуть результат

```
private static final int PICK_CONTACT_SUBACTIVITY = 2;

Uri uri = Uri.parse("content://contacts/people");
Intent intent = new Intent(Intent.ACTION_PICK, uri);
startActivityForResult(intent, PICK_CONTACT_SUBACTIVITY);
```

Возвращение результатов. Когда дочерняя Активность готова вернуть результат, прежде чем завершать ее работу с помощью функции finish, вызовите метод setResult. Он принимает два параметра: результирующий код и сам результат, представленный в виде Намерения.

Результирующий код говорит о том, с каким результатом завершилась работа Активности, как правило, это либо Activity.RESULT_OK, либо Activity.RESULT_CANCELED. В некоторых случаях нужно использовать собственный код возврата для обработки специфических для вашего приложения вариантов. Метод setResult поддерживает любое целочисленное значение.

Намерение, возвращаемое в качестве результата, часто включает путь URI к части содержимого (например, к выбранному контакту, телефонному номеру или медиафайлу) и набор значений в параметре extras, используемых для передачи дополнительной информации.

Листинг 5.5 представляет собой часть метода onCreate из дочерней Активности и показывает, как кнопки OK и Cancel могут возвращать в родительский компонент приложения разные результаты.

Листинг 5.5. Создание новых Общих настроек

```
Button okButton = (Button) findViewById(R.id.ok_button);
okButton.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    Uri data = Uri.parse("content://horses/" + selected_horse_id);

    Intent result = new Intent(null, data);
```

Продолжение ⇗

Листинг 5.5 (продолжение)

```
        result.putExtra(IS_INPUT_CORRECT, inputCorrect);
        result.putExtra(SELECTED_PISTOL, selectedPistol);
        setResult(RESULT_OK, result);
        finish();
      }
    });

    Button cancelButton = (Button) findViewById(R.id.cancel_button);
    cancelButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        setResult(RESULT_CANCELED, null);
        finish();
      }
    });
```

Если Активность была закрыта пользователем при нажатии аппаратной кнопки возврата или если метод finish был вызван раньше, чем setResult, результирующий код установится в RESULT_CANCELED, а возвращенное Намерение покажет значение null.

Обработка результатов, возвращаемых из дочерних Активностей. При закрытии дочерней Активности внутри родительского компонента срабатывает обработчик onActivityResult. Переопределите данный метод, чтобы обрабатывать результаты, возвращаемые дочерней Активностью.

Обработчик onActivityResult принимает несколько параметров.

- **Код запроса.** Код, который использовался для запуска Активности, возвращающей результат.
- **Результирующий код.** Код результата, устанавливаемый дочерней Активностью и указывающий, как завершилась ее работа. Это может быть любое целочисленное значение, но, как правило, либо Activity.RESULT_OK, либо Activity.RESULT_CANCELED.

ПРИМЕЧАНИЕ

Если работа дочерней Активности завершилась непредвиденно или если перед ее закрытием не был указан код результата, этот параметр станет равен Activity.RESULT_CANCELED.

- **Данные.** Намерение, используемое для упаковки возвращаемых данных. В зависимости от назначения дочерней Активности оно может включать путь URI, представляющий выбранную часть содержимого. В качестве альтернативы (или дополнения) дочерняя Активность может возвращать информацию в виде простых значений, упакованных в параметр Намерения extras.

В листинге 5.6 показан каркас для реализации обработчика событий onActivityResult, размещенного внутри Активности.

Листинг 5.6. Реализация обработчика onActivityResult

```
private static final int SHOW_SUB_ACTIVITY_ONE = 1;
private static final int SHOW_SUB_ACTIVITY_TWO = 2;

@Override
public void onActivityResult(int requestCode,
                             int resultCode,
                             Intent data) {

  super.onActivityResult(requestCode, resultCode, data);

  switch(requestCode) {
    case (SHOW_SUB_ACTIVITY_ONE) : {
      if (resultCode == Activity.RESULT_OK) {
        Uri horse = data.getData();
        boolean inputCorrect = data.getBooleanExtra(IS_INPUT_CORRECT,
false);
        String selectedPistol = data.getStringExtra(SELECTED_PISTOL);
      }
      break;
    }
    case (SHOW_SUB_ACTIVITY_TWO) : {
      if (resultCode == Activity.RESULT_OK) {
        // TODO: Обработка нажатия кнопки OK.
      }
      break;
    }
  }
}
```

Стандартные действия в Android

Системные приложения в Android также используют Намерения для запуска Активностей (обычных и дочерних).

В следующем списке, не претендующем на полноту, описываются некоторые стандартные действия, доступные в виде статических строковых констант из класса Intent. При создании неявных Намерений вы можете использовать данные действия для запуска обычных и дочерних Активностей внутри своих приложений.

ПРИМЕЧАНИЕ

Позже вы познакомитесь с Фильтрами намерений и научитесь регистрировать собственные Активности в виде обработчиков для этих действий.

- ACTION_ANSWER. Открывает Активность, которая связана с входящими звонками. На сегодняшний день это действие обрабатывается стандартным экраном для приема звонков.
- ACTION_CALL. Вызывает приложение для дозвона и немедленно инициирует звонок, используя номер, хранящийся в пути URI в Намерении.

В целом, если это возможно, рекомендуется использовать действие ACTION_DIAL.

- ACTION_DELETE. Запускает Активность, с помощью которой можно удалить данные, указанные в пути URI внутри Намерения.
- ACTION_DIAL. Открывает приложение для дозвона, содержащее уже набранный номер телефона из пути URI. По умолчанию, это действие обрабатывается стандартной программой для набора номера в Android. Она может нормализовать большинство форм записи: например и tel:555–1234, и tel:(212) 555 1212 — допустимые номера.
- ACTION_EDIT. Вызывает Активность, с помощью которой можно редактировать данные, находящиеся по указанному в Намерении пути URI.
- ACTION_INSERT. Открывает Активность для вставки в Курсор (Cursor) нового элемента, указанного с помощью пути URI. Дочерняя Активность, вызванная с этим действием, должна вернуть URI, ссылающийся на вставленный элемент.
- ACTION_PICK. Загружает дочернюю Активность, позволяющую выбрать элемент из Источника данных, указанный с помощью пути URI. При закрытии должен возвращаться URI, ссылающийся на выбранный элемент. Активность, которая будет запущена, зависит от типа выбранных данных, например при передаче пути `content://contacts/people` вызовется системный список контактов.
- ACTION_SEARCH. Запускает Активность для выполнения поиска. Поисковый запрос хранится в виде строки в дополнительном параметре Намерения по ключу SearchManager.QUERY.
- ACTION_SENDTO. Открывает Активность для отправки сообщений контакту, указанному в пути URI, который передается через Намерение.
- ACTION_SEND. Загружает экран для отправки данных, указанных в Намерении. Контакт-получатель должен быть выбран с помощью полученной Активности. Используйте метод setType, чтобы указать тип MIME для передаваемых данных.

  Эти данные должны храниться в параметре Намерения extras с ключами EXTRA_TEXT или EXTRA_STREAM, в зависимости от типа. В случае с электронной почтой стандартное приложение в Android также принимает дополнительные параметры по ключам EXTRA_EMAIL, EXTRA_CC, EXTRA_BCC и EXTRA_SUBJECT. Используйте действие ACTION_SEND только в тех случаях, когда данные нужно передать удаленному адресату (а не другой программе на том же устройстве).

- ACTION_VIEW. Наиболее распространенное общее действие. Для данных, передаваемых с помощью пути URI в Намерении, ищется наиболее подходящий способ вывода. Выбор приложения зависит от схемы (протокола) данных. Стандартные адреса http: будут открываться в браузере, адреса tel: — в приложении для дозвона, geo: — в программе Google Maps, а данные о контакте — отображатся в приложении для управления контактной информацией.
- ACTION_WEB_SEARCH. Открывает Активность, которая ведет поиск в Интернете, основываясь на тексте, переданном с помощью пути URI (как правило, при этом запускается браузер).

ПРИМЕЧАНИЕ

Помимо выше представленных действий для Активностей, Android включает большое количество широковещательных действий, используемых для создания Намерений, которые транслируют уведомления о системных событиях. Данные действия описаны далее в этой главе.

Использование тегов intent-filter для обслуживания неявных Намерений

Если Намерение запрашивает выполнение какого-либо действия с определенным набором данных, каким образом Android выбирает приложение (или компонент) для обслуживания этого запроса?

Фильтры намерений используются для регистрации Активностей, Сервисов и Широковещательных приемников в качестве компонентов, способных выполнять заданные действия с конкретным видом данных. С помощью этих Фильтров также регистрируются Широковещательные приемники, настроенные на трансляцию Намерением заданного действия или события.

Задействуя Фильтры намерений, приложения объявляют, что они могут отвечать на действия, запрашиваемые любой другой программой, установленной на устройстве.

Чтобы зарегистрировать компонент приложения в качестве потенциального обработчика Намерений, добавьте тег `intent-filter` в узел компонента в манифесте. Внутри узла Фильтра намерений можно использовать такие теги (и соответствующие атрибуты).

- `action`. Использует атрибут `android:name`, чтобы задать название действия, которое будет обслуживаться. Каждый Фильтр намерений должен иметь один (и только один) тег `action`. Действия должны быть представлены в виде уникальных строк, которые сами себя описывают. Рекомендуется выбирать названия, основываясь на соглашении об именовании пакетов в Java.

- category. Использует атрибут android:name, чтобы указать, при каких обстоятельствах должно обслуживаться действие. Каждый тег intent-filter способен содержать несколько тегов category. Вы можете задать собственные категории или же брать стандартные значения, предоставляемые системой.
  - ALTERNATIVE. Наличие данной категории говорит о том, что действие должно быть доступно в качестве альтернативного тому, которое выполняется по умолчанию для элемента этого типа данных. Например, если действие по умолчанию для контакта — просмотр, то в качестве альтернативы его также можно редактировать.
  - SELECTED_ALTERNATIVE. То же самое, что и ALTERNATIVE, но вместо одиночного действия с использованием утверждения Намерения, которое описано выше, применяется в тех случаях, когда нужен список различных возможностей. Одной из функций Фильтра намерений может стать динамическое заполнение контекстного меню с помощью действий. Данный процесс продемонстрирован далее в этой главе.
  - BROWSABLE. Говорит о том, что действие доступно из браузера. Когда Намерение срабатывает в браузере, оно всегда содержит данную категорию. Если вы хотите, чтобы приложение реагировало на действия, инициированные браузером (такие как перехват ссылок на конкретный сайт), то должны добавить в его манифест категорию BROWSABLE.
  - DEFAULT. Установите эту категорию, чтобы сделать компонент обработчиком по умолчанию для действия, выполняемого с указанным типом данных внутри Фильтра намерений. Это необходимо и для Активностей, которые запускаются с помощью явных Намерений.
  - GADGET. Наличие этой категории указывает на то, что данная Активность может запускаться внутри другой Активности.
  - HOME. Устанавливая эту категорию и не указывая при этом действия, вы создаете альтернативу для стандартного домашнего экрана.
  - LAUNCHER. Используя эту категорию, вы помещаете Активность в окно для запуска приложений.
- data. Этот тег дает возможность указать тип данных, с которым может взаимодействовать ваш компонент. При необходимости можно задать несколько тегов data. Чтобы указать, какие именно данные поддерживает ваш компонент, используйте сочетание следующих атрибутов:
  - android:host — задает доступное имя удаленного сервера (например, google.com);

- android:mimetype — позволяет указать тип данных, которые ваш компонент способен обрабатывать. Для примера: <type android: value="vnd.android.cursor.dir/*"/> будет соответствовать любому Курсору в Android;
- android:path — задает доступные значения для пути URI (например, /transport/boats/);
- android:port — указывает доступные порты для заданного сервера;
- android:scheme — требует указать конкретную схему (например, content или http).

В листинге 5.7 показан Фильтр намерений для Активности, которая может выполнять SHOW_DAMAGE либо как главное, либо как альтернативное действие (данные о землетрясениях вы научитесь получать в следующей главе).

Листинг 5.7. Регистрация Активности в качестве Приемника намерений

```
<activity android:name=".EarthquakeDamageViewer" android:label="View
Damage">
  <intent-filter>
    <action android:name="com.paad.earthquake.intent.action.SHOW_
DAMAGE"></action>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.SELECTED_
ALTERNATIVE"/>
    <data android:mimeType="vnd.earthquake.cursor.item/*"/>
  </intent-filter>
</activity>
```

Принцип работы Фильтров намерений в Android

При использовании метода startActivity передаваемое неявное Намерение, как правило, доходит лишь до одной Активности. Если для выполнения заданного действия с указанными данными годятся сразу несколько Активностей, пользователю предоставлен список выбора.

Процесс, когда решается, какую Активность лучше запустить, называется Утверждением намерений. Его цель — найти наиболее подходящий Фильтр намерений. В целом весь алгоритм можно описать следующим образом.

1. Android собирает список всех доступных Фильтров намерений из установленных пакетов.
2. Фильтры, которые не соответствуют действию или категории Намерения, удаляются из списка.

   Совпадение происходит только в том случае, если Фильтр намерений содержит указанное действие (или если действие для него вовсе не задано).

Совпадения не произойдет, только если ни одно из действий Фильтра намерений не будет эквивалентно тому, которое задано в Намерении.

Для категорий процесс соответствия более строгий. Фильтр намерений должен включать в себя все категории, заданные в полученном Намерении. Фильтр, для которого категории не указаны, может соответствовать только таким же Намерениям (нет категорий).

3. Наконец, каждая часть пути URI из Намерения сравнивается с тегом `data` Фильтра намерений. Если в Фильтре указаны схема (протокол), сервер/принадлежность, путь или тип MIME, все эти значения проверяются на соответствие пути URI из Намерения. При любом несовпадении Фильтр будет удален из списка. Если в Фильтре намерений не указано ни одного параметра `data`, его действие будет распространяться на любые данные.

   - MIME — тип данных, который должен совпасть. При сравнении типов данных вы можете использовать маски, чтобы охватывать все подтипы (например, `earthquakes/*`). Если в Фильтре намерения указан тип данных, он должен совпасть с тем, который значится в намерении, при отсутствии тега `data` подойдет любой тип.
   - Схема — это протокольная часть пути URI, например `http:`, `mailto:` или `tel:`.
   - Имя сервера (или принадлежность данных) — часть URI между схемой и самим путем (например, www.google.com). Чтобы совпало имя сервера, схема Фильтра намерений также должна подойти.
   - После имени сервера идет путь к данным (например, `/ig`). Путь пройдет проверку только после схемы и имени сервера, содержащихся в теге.

4. Когда вы неявным образом запускаете Активность и вышеописанный процесс возвращает более одного совпадения, пользователю выводится список со всеми вариантами.

Компоненты стандартных приложений в Android участвуют в процессе утверждения Намерений точно так же, как и сторонние программы. Они не имеют повышенного приоритета и могут быть полностью заменены новыми Активностями, которые содержат Фильтры намерений для соответствующих действий.

Поиск и использование Намерения, с помощью которого была запущена Активность

Когда программный компонент запускается с помощью неявного Намерения, он должен найти действие, которое необходимо осуществить, и данные для него.

Используйте метод getIntent (как правило, внутри метода onCreate), чтобы извлечь Намерение, с помощью которого компонент запущен. Этот процесс показан в листинге 5.8.

Листинг 5.8. Поиск исходного Намерения в дочерней Активности

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  setContentView(R.layout.main);

  Intent intent = getIntent();
}
```

Применяйте методы getAction и getData, чтобы найти действие и данные, связанные с Намерением. Для извлечения дополнительной информации, хранящейся в параметре extras, используйте типизированные методы `get<тип>Extra`.

```
String action = intent.getAction();
Uri data = intent.getData();
```

Делегирование ответственности

Как показано в листинге 5.9, задействуйте метод startNextMatchingActivity для делегирования ответственности за обработку действия следующему компоненту, который лучше всего для этого подходит.

Листинг 5.9. Передача обязанностей по обработке действия другому Приемнику намерений

```
Intent intent = getIntent();
if (isDuringBreak)
  startNextMatchingActivity(intent);
```

Это позволяет добавлять дополнительные условия для компонентов, которые ограничат сферу их применения рамками утверждения Намерений.

В некоторых случаях ваш компонент может выполнить некоторые операции или предложить пользователю выбор, прежде чем передать Намерение другому компоненту.

Пример приложения для выбора контактов

В этом примере[1] вы создадите новую Активность, которая обслуживает действие ACTION_PICK для контактных данных. Она будет показывать все контакты из базы данных и давать пользователю возможность выбрать

[1] Все фрагменты кода в данном примере — часть проекта Contact Picker из главы 5, их можно загрузить с сайта Wrox.com.

один из них, после чего закроется и вернет родительской Активности путь URI к выбранному контакту.

ПРИМЕЧАНИЕ

Данный пример немного надуманный. Android уже содержит Фильтр намерений для выбора контакта из списка, который может предоставлять URI вида content://contacts/people/ внутри неявного Намерения. Цель этого учебного проекта состоит в демонстрации общих принципов, даже если пользы от этой конкретной реализации немного.

1. Создайте новый проект ContactPicker, который содержит одноименную Активность.

```
package com.paad.contactpicker;

import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Contacts.People;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.AdapterView.OnItemClickListener;

public class ContactPicker extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
  }
}
```

2. Отредактируйте ресурс с разметкой `main.xml`, добавив в него элемент ListView для отображения контактов.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <ListView android:id="@+id/contactListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
</LinearLayout>
```

3. Создайте новый ресурс с разметкой `listitemlayout.xml`, содержащий единственный элемент TextView. Он нужен для отображения каждого отдельного контакта.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <TextView
    android:id="@+id/itemTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="10px"
    android:textSize="16px"
    android:textColor="#FFF"
  />
</LinearLayout>
```

4. Вернитесь к Активности ContactPicker. Переопределите метод onCreate, чтобы извлечь путь к данным из Намерения, которое эту Активность вызвало.

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  setContentView(R.layout.main);

  Intent intent = getIntent();
  String dataPath = intent.getData().toString();
```

   4.1. Создайте новый путь URI для данных о людях, которые хранятся в списке контактов, и свяжите его с элементом ListView с помощью класса SimpleCursorAdapter:

ПРИМЕЧАНИЕ

Класс SimpleCursorAdapter позволяет назначать данные Курсора, используемые Источником данных, для Представлений. Здесь он приводится без дополнительных комментариев, но позже в этой главе вы рассмотрите его более подробно.

```
final Uri data = Uri.parse(dataPath + "people/");
final Cursor c = managedQuery(data, null, null, null);

String[] from = new String[] {People.NAME};
int[] to = new int[] { R.id.itemTextView };

SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
                                                      R.layout.
listitemlayout,
                                                      c,
                                                      from,
                                                      to);
ListView lv = (ListView)findViewById(R.id.contactListView);
lv.setAdapter(adapter);
```

4.2. Добавьте обработчик onItemClickListener для ListView. При выборе контакта из списка в родительскую Активность должен возвращаться путь к выбранному элементу:

```
lv.setOnItemClickListener(new OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent, View view, int pos,
                         long id) {
    // Переместите Курсор к выбранному элементу.
    c.moveToPosition(pos);
    // Извлеките идентификатор строки.
    int rowId = c.getInt(c.getColumnIndexOrThrow("_id"));
    // Соберите воедино результирующий путь URI.
    Uri outURI = Uri.parse(data.toString() + rowId);
    Intent outData = new Intent();
    outData.setData(outURI);
    setResult(Activity.RESULT_OK, outData);
    finish();
  }
});
```

4.3. Закройте метод onCreate:

```
}
```

5. Отредактируйте манифест приложения, заменив тег `intent-filter` для Активности и добавив поддержку действия ACTION_PICK, применимого к контактным данным:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.paad.contactpicker">
  <application android:icon="@drawable/icon">
    <activity android:name="ContactPicker" android:label="@string/app_
name">
      <intent-filter>
        <action android:name="android.intent.action.PICK"></action>
        <category android:name="android.intent.category.DEFAULT"></ca-
tegory>
        <data android:path="contacts" android:scheme="content"></data>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

6. На этом заканчивается работа с дочерней Активностью. Чтобы проверить ее, создайте новую тестовую Активность ContentPickerTester, новый ресурс для разметки `contentpickertester.xml`, содержащий элемент TextView для отображения выбранного контакта, а также кнопку Button, чтобы запустить дочернюю Активность.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
```

```
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <TextView
    android:id="@+id/selected_contact_textview"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
  <Button
    android:id="@+id/pick_contact_button"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Pick Contact"
  />
</LinearLayout>
```

7. Переопределите метод onCreate из класса ContentPickerTester, добавив в него реализацию OnClickListener для элемента Button, чтобы при нажатии запускалась новая дочерняя Активность с действием ACTION_PICK и путем URI к базе данных, содержащей контактную информацию (`content://contacts/`).

```
package com.paad.contactpicker;

import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Contacts.People;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class ContentPickerTester extends Activity {

  public static final int PICK_CONTACT = 1;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.contentpickertester);

    Button button = (Button)findViewById(R.id.pick_contact_button);

    button.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View _view) {
        Intent intent = new Intent(Intent.ACTION_PICK,
                            Uri.parse("content://contacts/"));
        startActivityForResult(intent, PICK_CONTACT);
      }
    });
  }
}
```

8. Получив результат из дочерней Активности, используйте его для отображения имени выбранного контакта с помощью элемента TextView.

```
@Override
public void onActivityResult(int reqCode, int resCode, Intent data) {
  super.onActivityResult(reqCode, resCode, data);

  switch(reqCode) {
    case (PICK_CONTACT) : {
      if (resCode == Activity.RESULT_OK) {
        Uri contactData = data.getData();
        Cursor c = managedQuery(contactData, null, null, null);
        c.moveToFirst();
        String name = c.getString(c.getColumnIndexOrThrow(People.NAME));
        TextView tv = (TextView)findViewById(R.id.selected_contact_
textview);
        tv.setText(name);
      }
      break;
    }
  }
}
```

9. Закончив с тестовой Активностью, внесите ее в манифест своего приложения. Вам также необходимо добавить полномочие READ_CONTACTS внутри тега `uses-permission`, чтобы открыть приложению доступ к базе данных контактов.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.paad.contactpicker">
  <application android:icon="@drawable/icon">
    <activity android:name=".ContactPicker" android:label="@string/app_
name">
      <intent-filter>
        <action android:name="android.intent.action.PICK"></action>
        <category android:name="android.intent.category.DEFAULT"></
category>
        <data android:path="contacts" android:scheme="content"></data>
      </intent-filter>
    </activity>
    <activity android:name=".ContentPickerTester"
              android:label="Contact Picker Test">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
</manifest>
```

Запустите Активность и нажмите кнопку. При этом должна появиться Активность ContactPicker, как показано на рис. 5.1.

После того, как выберете контакт, родительская Активность должна отобразить его имя, выйдя на передний план, как показано на рис. 5.2.

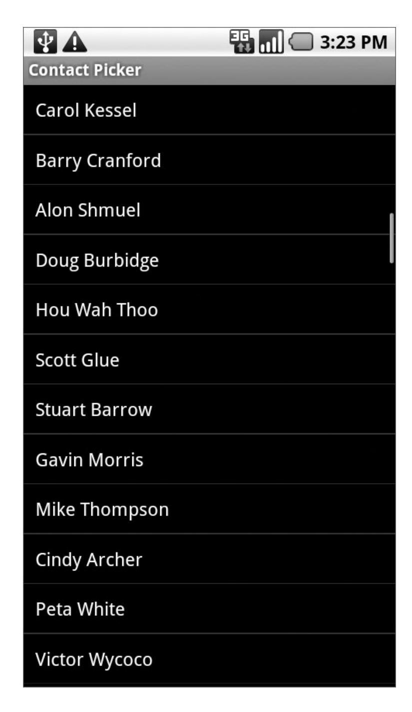

3:23 PM
Contact Picker
Carol Kessel
Barry Cranford
Alon Shmuel
Doug Burbidge
Hou Wah Thoo
Scott Glue
Stuart Barrow
Gavin Morris
Mike Thompson
Cindy Archer
Peta White
Victor Wycoco

Рис. 5.1.

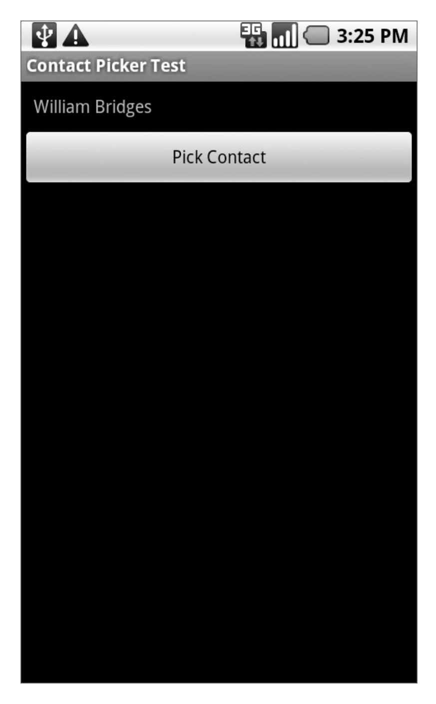

3:25 PM
Contact Picker Test
William Bridges
Pick Contact

Рис. 5.2.

Использование Фильтров намерений для создания дополнений и обеспечения расширяемости

Теперь вы знаете, как создавать неявные Намерения для запуска Активностей, но самое интересное еще впереди.

Кроме всего прочего, Android позволяет новым пакетам расширять функции уже существующих приложений, используя Фильтры намерений, чтобы динамически, во время выполнения программы, добавлять новые пункты меню.

Благодаря этому вы можете создавать для Активностей подключаемые модули, получая потенциальную выгоду от новой функциональности, которую могут в себе нести еще не созданные программные компоненты. Таким образом, не придется изменять или заново собирать свои проекты.

Метод addIntentOptions из класса Menu позволяет указывать Намерение, описывающее данные для построения меню. Android принимает это Намерение и возвращает все действия, заданные для Фильтра намерений, которые отвечают указанным данным. Для каждого действия создается объект MenuItem, содержащий текст из соответствующей метки Фильтра.

Изящность такой концепции проще всего объяснить на примере. Если данные вашей Активности отображаются в виде списка мест, контекстное меню для каждого элемента должно содержать пункты Показать и Отобразить маршрут. Заглянем на несколько лет вперед: вы создали приложение, которое взаимодействует с вашим автомобилем, позволяя управлять им с помощью телефона. Поскольку меню создается динамически, когда новый Фильтр намерений, содержащий действие DRIVE_CAR, добавляется в узел Активности в манифесте, Android автоматически включает это действие в ваше приложение в виде объекта MenuItem.

Возможность динамического заполнения меню позволяет модифицировать функциональность приложения при создании нового компонента, способного выполнять действия с определенным видом данных. Многие системные приложения в Android используют эту возможность, разрешая вносить новые действия в стандартные Активности.

Добавление анонимных действий в приложения

Чтобы использовать механизм, с помощью которого действия из вашей Активности могут стать доступными анонимно для имеющихся приложений, опубликуйте их, используя теги `intent-filter` внутри соответствующих узлов в манифесте.

Фильтры намерений описывают выполняемые действия и данные для них. Именно данные будут задействованы в процессе утверждения Намерений для определения, когда должно стать доступным конкретное действие. Тег `category` нужно установить либо в ALTERNATIVE, либо в SELECTED_

ALTERNATIVE (может присутствовать сразу два тега с обоими значениями). Текст, используемый в элементах меню, указывается с помощью атрибута `android:label`.

В листинге 5.10 показан пример Фильтра намерений, который декларирует способность Активности сбивать с орбиты лунные базы с помощью ядерных ракет.

Листинг 5.10. Объявление о действиях, поддерживаемых Активностью

```
<activity android:name=".NostromoController">
  <intent-filter
    android:label="Nuke From Orbit">
    <action android:name="com.pad.nostromo.NUKE_FROM_ORBIT" />
    <data android:mimeType="vnd.moonbase.cursor.item/*"/>
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
  </intent-filter>
</activity>
```

Источник данных и другие части кода, необходимые для запуска этого примера, здесь не представлены. В следующих разделах вы узнаете, как динамически добавлять действия для меню из другой Активности.

Добавление анонимных действий в меню вашей Активности

Чтобы динамически, во время выполнения программы, добавить экземпляры MenuItem в меню, используйте метод addIntentOptions из объекта Menu (передайте Намерение, определяющее данные, для которых вы хотите предоставить действия). Как правило, все эти манипуляции происходят внутри обработчиков onCreateOptionsMenu и onCreateContextMenu вашей Активности.

Созданное Намерение послужит для нахождения компонентов, содержащих теги `intent-filter`, которые поддерживают действия с указанными вами данными. В связи с этим Намерение не должно содержать собственных действий, только данные для них. Необходимо также указать категорию для действия: CATEGORY_ALTERNATIVE или CATEGORY_SELECTED_ALTERNATIVE.

Процесс создания Намерения для меню следующий:

```
Intent intent = new Intent();
intent.setData(MyProvider.CONTENT_URI);
intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
```

Передайте это Намерение в меню, которое хотите заполнить, используя метод addIntentOptions. Вы также можете задать опциональные флаги, имя вызывающего класса, группу и идентификатор меню. Кроме того, есть возможность указать массив Намерений, которые вы хотите применить для создания дополнительных пунктов меню.

Из листинга 5.11 вы можете получить представление о том, как динамически формировать меню для Активности, добавляя туда действие NUKE_FROM_ORBIT, описанное в предыдущем листинге.

Листинг 5.11. Динамическое формирование меню с помощью действий Активности

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  // Создайте Намерение, с помощью которого будут получены
  // действия, необходимые для формирования меню.
  Intent intent = new Intent();
  intent.setData(MoonBaseProvider.CONTENT_URI);
  intent.addCategory(Intent.CATEGORY_SELECTED_ALTERNATIVE);

  // Стандартные свойства меню, позволяющие вам задать группу и
  // идентификатор для пунктов, которые вы хотите добавить.
  int menuGroup = 0;
  int menuItemId = 0;
  int menuItemOrder = Menu.NONE;

  // Предоставьте имя компонента, который вызывает действие -
  // как правило, это текущая Активность.
  ComponentName caller = getComponentName();

  // Сперва определите Намерения, которые должны быть добавлены.
  Intent[] specificIntents = null;
  // Пункты меню, созданные из предыдущих Намерений,
  // заполнят этот массив.
  MenuItem[] outSpecificItems = null;

  // Установите опциональные флаги.
  int flags = Menu.FLAG_APPEND_TO_GROUP;

  // Сформируйте меню
  menu.addIntentOptions(menuGroup,
                        menuItemId,
                        menuItemOrder,
                        caller,
                        specificIntents,
                        intent,
                        flags,
                        outSpecificItems);

  return true;
}
```

Введение в Linkify

Linkify — вспомогательный класс, который автоматически создает гиперссылки внутри элементов TextView (и их производных) на основе регулярных выражений RegEx.

Текст, соответствующий заданному регулярному выражению, будет преобразован в гиперссылку, при нажатии которой срабатывает код вида `startActivity` (new Intent(Intent.ACTION_VIEW, uri)), где uri — совпавший текст.

Вы можете задать шаблон строк, которые хотите превратить в ссылки. Для удобства класс Linkify предлагает предустановленные шаблоны для часто встречающихся типов содержимого (таких как телефонные номера или адреса электронной почты), которые описаны в следующем разделе.

Стандартные типы ссылок в Linkify

Статический метод Linkify.addLinks в качестве параметров принимает Представление, к которому будут применены шаблоны, а также битовые маски для одного или нескольких стандартных типов содержимого, предоставляемые классом Linkify: WEB_URLS, EMAIL_ADDRESSES, PHONE_NUMBERS и ALL.

В листинге 5.12 показано, как с помощью класса Linkify отобразить адреса веб-сайтов и электронной почты, содержащихся в элементе TextView, в виде гиперссылок. При нажатии ссылки откроются браузер и почтовый клиент соответственно.

Листинг 5.12. Использование Linkify в коде программы

```
TextView textView = (TextView)findViewById(R.id.myTextView);
Linkify.addLinks(textView, Linkify.WEB_URLS|Linkify.EMAIL_ADDRESSES);
```

ПРИМЕЧАНИЕ

Большинство устройств под управлением Android включают как минимум два почтовых приложения: Gmail и Email. Если для выполнения действия подходят сразу несколько Активностей, пользователю предложено выбрать между ними.

Вы также можете применить класс Linkify к Представлению прямо внутри ресурса с разметкой, используя атрибут `android:autoLink`. Он поддерживает одно или несколько значений (разделенных символом |), названия которых говорят сами за себя: none, web, email, phone и all.

В листинге 5.13 показано, как добавить гиперссылку для телефонных номеров и адресов электронной почты.

Листинг 5.13. Использование Linkify с помощью XML

```
<TextView
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:text="@string/linkify_me"
  android:autoLink="phone|email"
/>
```

Создание собственных шаблонов для Linkify

Чтобы описать собственный шаблон для Linkify, необходимо создать новое регулярное выражение RegEx, соответствующее тексту, который вы хотите отобразить в виде ссылки.

Как и в случае со стандартными типами, применять Linkify к Представлению можно с помощью метода Linkify.addLinks, но вместо константы необходимо передать новое регулярное выражение RegEx. Вы также можете передать префикс, который будет добавлен к целевому пути URI после нажатия ссылки.

В листинге 5.14 показано, как применить Linkify к Представлению для добавления поддержки данных о землетрясениях, предоставляемых с помощью Источника данных (который вы создадите в главе 7). Чтобы не добавлять всю схему, шаблон для Linkify считает за совпадение любой текст, начинающийся со строки quake, за которым следует номер. Потом схема добавляется в начало пути URI, прежде чем сработает Намерение.

Листинг 5.14. Создание собственных шаблонов для Linkify

```
int flags = Pattern.CASE_INSENSITIVE;
Pattern p = Pattern.compile("\\bquake[0-9]*\\b", flags);
Linkify.addLinks(myTextView, p,
                 "content://com.paad.earthquake/earthquakes/");
```

Linkify также поддерживает интерфейсы TransformFilter и MatchFilter, что позволяет установить дополнительный контроль над структурой целевого пути URI и описание регулярного выражения. Пример использования этих интерфейсов представлен в следующем фрагменте кода:

```
Linkify.addLinks(myTextView, pattern, prefixWith,
                 new MyMatchFilter(), new MyTransformFilter());
```

Использование интерфейса MatchFilter

Реализуйте метод acceptMatch в своем классе, который унаследован от MatchFilter, чтобы добавить дополнительные условия в регулярное выражение. acceptMatch срабатывает при нахождении потенциального совпадения, принимая в качестве параметров начальный и конечный индексы (вместе с полной искомой строкой).

В листинге 5.15 показана реализация интерфейса MatchFilter, которая отменяет любое совпадение, если перед найденной строкой находится знак восклицания.

Листинг 5.15. Использование MatchFilter для Linkify

```
class MyMatchFilter implements MatchFilter {
  public boolean acceptMatch(CharSequence s, int start, int end) {
    return (start == 0 || s.charAt(start-1) != '!');
  }
}
```

Использование интерфейса TransformFilter

TransformFilter дает больше свободы для форматирования текстовых строк, изменения внешнего вида генерируемых ссылок. Отделение текста ссылки от целевого пути URI позволяет менять способ вывода текстовых данных на экраны.

Чтобы задействовать интерфейс TransformFilter, необходимо переопределить метод transformUrl. Он начнет вызываться при нахождении совпадения. В качестве параметров он принимает регулярное выражение и строку URI, которую должен создать. Вы можете изменить совпавшую строку и вернуть путь URI в качестве цели, подходящей для отображения с помощью другой программы.

Реализация интерфейса TransformFilter, приведенная в листинге 5.16, преобразует совпавшую строку в путь URI, возвращаемый в нижнем регистре.

Листинг 5.16. Использование TransformFilter для Linkify

```
class MyTransformFilter implements TransformFilter {
  public String transformUrl(Matcher match, String url) {
    return url.toLowerCase();
  }
}
```

Использование Намерений для трансляции событий

Намерения, будучи системным механизмом для передачи сообщений, способны отправлять структурированные данные от процесса к процессу.

До сих пор вы использовали Намерения для запуска новых программных компонентов, но с их помощью также можно анонимно передавать сообщения между разными частями приложений, вызывая метод sendBroadcast. В своих программах вы можете реализовать Широковещательные приемники для отслеживания этих Намерений и ответа на них.

Широковещательные намерения используются для объявления о системных или прикладных событиях, выводя событийную программную модель за рамки единственного приложения.

Трансляция Намерений помогает сделать приложение более открытым. Передавая событие с Намерением, вы получаете возможность (и предоставляете ее другим разработчикам) реагировать на него без необходимости модифицирования оригинальной программы. Внутри своего приложения вы можете отслеживать Широковещательные намерения, чтобы заменить или улучшить стандартные (или сторонние) компоненты, а также реагировать на системные изменения и события в других приложениях.

Широковещательные намерения широко используются в Android для трансляции таких системных событий, как уровень заряда батареи, состояние подключения к сети и входящие звонки.

Трансляция событий с помощью Намерений

Трансляция Намерений — процесс простой. Создайте внутри своего приложения Намерение, которое хотите передать, и используйте метод sendBroadcast для отправки.

Укажите действие, данные и категорию для вашего Намерения таким образом, чтобы Широковещательные приемники смогли без труда его распознать. В данном случае строка action нужна для определения транслируемого события, поэтому она должна быть уникальной. Такие строки принято составлять по тому же принципу, что и имена для пакетов в Java:

```
public static final String NEW_LIFEFORM_DETECTED =
  "com.paad.action.NEW_LIFEFORM";
```

Если вы захотите включить какие-либо данные в Намерение, то можете указать путь URI, используя свойство data. Вы также вправе добавлять дополнительные параметры для простых значений. Согласно событийной парадигме, содержимое extras приравнивается к необязательным параметрам и передается в обработчик события.

В листинге 5.17 показаны основные этапы создания Широковещательного намерения с использованием описанного ранее действия и включением дополнительной информации, которая хранится в виде параметра extras.

Листинг 5.17. Трансляция Намерения

```
Intent intent = new Intent(NEW_LIFEFORM_DETECTED);
intent.putExtra("lifeformName", lifeformType);
intent.putExtra("longitude", currentLongitude);
intent.putExtra("latitude", currentLatitude);
sendBroadcast(intent);
```

Отслеживание трансляций с помощью Приемников широковещательных намерений

Широковещательные приемники используются для отслеживания транслируемых Намерений. Чтобы Приемник включить, его нужно зарегистрировать либо в коде, либо в манифесте приложения. При регистрации Широковещательного приемника нужно использовать Фильтр намерений для указания объекта Intent, за которым требуется следить.

Чтобы создать новый Широковещательный приемник, наследуйте класс BroadcastReceiver и переопределите обработчик событий onReceive, как показано в листинге 5.18.

Листинг 5.18. Каркас для реализации BroadcastReceiver

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class MyBroadcastReceiver extends BroadcastReceiver {
  @Override
```

```
  public void onReceive(Context context, Intent intent) {
    //TODO: Реакция на полученное Намерение.
  }
}
```

Метод onReceive будет выполняться, если полученное Намерение соответствует Фильтру, который использовался для регистрации Приемника. Этот метод должен быть завершен в течение пяти секунд, иначе появится диалоговое окно с информацией о принудительном закрытии.

Приложения, содержащие в своем манифесте Широковещательный приемник, не обязаны выполняться во время трансляции Намерения, чтобы его получить. Они запускаются автоматически при передаче соответствующего объекта Intent. Такой подход хорош при управлении ресурсами, так как позволяет создавать приложения, которые используют событийную модель и отвечают на транслируемые события даже после закрытия или преждевременного завершения.

Как правило, Приемники широковещательных намерений применяются для обновления содержимого и пользовательского интерфейса Активностей, запуска Сервисов или уведомления пользователя с помощью объекта NotificationManager. Пятисекундное ограничение гарантирует, что основные операции не могут (и не должны) выполняться внутри самого Широковещательного приемника.

В листинге 5.19 показана реализация Приемника широковещательных намерений. В следующих разделах вы научитесь регистрировать его как с помощью манифеста приложения, так и внутри кода программы.

Листинг 5.19. Реализация BroadcastReceiver

```
public class LifeformDetectedBroadcastReceiver extends BroadcastReceiver {

  public static final String BURN = "com.paad.alien.action.BURN_IT_WITH_
FIRE";

  @Override
  public void onReceive(Context context, Intent intent) {
    // Получите подробные данные из Намерения.
    Uri data = intent.getData();
    String type = intent.getStringExtra("type");
    double lat = intent.getDoubleExtra("latitude", 0);
    double lng = intent.getDoubleExtra("longitude", 0);
    Location loc = new Location("gps");
    loc.setLatitude(lat);
    loc.setLongitude(lng);
    if (type.equals("alien")) {
      Intent startIntent = new Intent(BURN, data);
      startIntent.putExtra("latitude", lat);
      startIntent.putExtra("longitude", lng);

      context.startActivity(startIntent);
    }
  }
}
```

Регистрация Широковещательных приемников внутри манифеста приложения. Чтобы включить Широковещательный приемник в манифест приложения, добавьте тег `receiver` внутрь узла `application`, указав соответствующее имя класса. Узел получателя должен содержать тег `intent-filter`, в нем нужно задать действие (в виде строки), которое необходимо отслеживать. Пример показан в листинге 5.20.

Листинг 5.20. Регистрация Широковещательного приемника с помощью XML

```
<receiver android:name=".LifeformDetectedBroadcastReceiver">
  <intent-filter>
    <action android:name="com.paad.action.NEW_LIFEFORM"/>
  </intent-filter>
</receiver>
```

Приемники, зарегистрированные таким образом, всегда остаются активными и будут получать Широковещательные намерения даже тогда, когда приложение не работает или преждевременно завершилось.

Регистрация Широковещательных приемников внутри программного кода. Вы также можете зарегистрировать Приемник широковещательных намерений внутри кода программы. Приемник, зарегистрированный таким способом, будет отвечать на транслируемые Намерения только в том случае, если компонент приложения, которому он принадлежит, выполняется в этот момент.

Как правило, это полезно, когда Приемник используется для обновления элементов пользовательского интерфейса внутри Активности. В этом случае рекомендуется отменять регистрацию Широковещательного приемника, если Активность не отображается на экране (или не находится в активном состоянии).

В листинге 5.21 показано, как зарегистрировать Широковещательный приемник внутри программного кода, используя класс IntentFilter.

Листинг 5.21. Регистрация Широковещательного приемника внутри кода программы

```
// Создайте и зарегистрируйте Широковещательный приемник.
IntentFilter filter = new IntentFilter(NEW_LIFEFORM_DETECTED);
LifeformDetectedBroadcastReceiver r = new
LifeformDetectedBroadcastReceiver();
registerReceiver(r, filter);
```

Чтобы отменить регистрацию, используйте метод unregisterReceiver в контексте приложения, передавая ему в качестве параметра экземпляр Широковещательного приемника, как показано ниже:

```
unregisterReceiver(receiver);
```

Другие примеры можно найти в главе 9. Там же вы узнаете, как создавать свои фоновые Сервисы и использовать Намерения, чтобы транслировать события для собственных Активностей.

Трансляция «липких» и упорядоченных Намерений

Если Намерение транслируется с помощью метода sendBroadcast, оно будет принято всеми Широковещательными приемниками, при этом вы не можете контролировать порядок распространения результатов.

В случаях, когда порядок получения Намерений Приемниками играет важную роль или когда нужно, чтобы Приемники получили возможность влиять на транслируемое Намерение, можете использовать метод sendOrderedBroadcast:

```
sendOrderedBroadcast(intent, null);
```

С помощью этого метода ваше Намерение получат все зарегистрированные Приемники в порядке, основанном на заданном приоритете. При необходимости вы также можете назначить собственный Широковещательный приемник, который будет получать Намерение уже после того, как оно обработается (и, возможно, модифицируется) всеми другими Приемниками:

```
sendOrderedBroadcast(intent, null, myBroadcastReceiver, null,
                     Activity.RESULT_OK, null, null);
```

Для большей эффективности некоторые трансляции делаются «липкими» (Sticky). Когда вы вызываете метод registerReceiver и передаете ему Фильтр намерений, отслеживающий такие трансляции, в качестве возвращаемого значения получите «липкое» Широковещательное намерение. Чтобы транслировать его, ваше приложение должно иметь полномочие BROADCAST_STICKY:

```
sendStickyBroadcast(intent);
```

Чтобы убрать «липкое» Намерение, передайте его в виде параметра в метод removeStickyBroadcast:

```
removeStickyBroadcast(intent);
```

Стандартные широковещательные действия в Android

Android транслирует Намерения для многих системных Сервисов. Вы можете использовать эти сообщения, чтобы добавить в собственные проекты новую функциональность, основанную на системных событиях, таких как изменение временной зоны и состояния подключения, получение SMS-сообщений или прием телефонных звонков.

В следующем перечне описываются некоторые стандартные действия, хранящиеся в виде констант в классе Intent. Эти действия в основном предназначены для отслеживания состояния устройства.

- ACTION_BOOT_COMPLETED. Срабатывает, как только устройство завершило запуск. Чтобы получать это действие, приложение должно иметь полномочие RECEIVE_BOOT_COMPLETED.
- ACTION_CAMERA_BUTTON. Срабатывает при нажатии кнопки камеры.
- ACTION_DATE_CHANGED и ACTION_TIME_CHANGED. Эти действия транслируются, если настройки даты или времени на устройстве изменены вручную (в отличие от изменений, связанных с естественным течением времени).
- ACTION_MEDIA_BUTTON. Срабатывает, если нажата мультимедийная кнопка.
- ACTION_MEDIA_EJECT. Если пользователь инициировал извлечение внешнего накопителя, это событие сработает в первую очередь. Когда приложение взаимодействует с внешним носителем данных, вы должны отслеживать это событие, чтобы сохранить и закрыть любой открытый файл.
- ACTION_MEDIA_MOUNTED и ACTION_MEDIA_UNMOUNTED. Эти два события транслируются каждый раз при добавлении или извлечении внешнего накопителя.
- ACTION_NEW_OUTGOING_CALL. Транслируется перед осуществлением исходящего звонка. Отслеживайте это событие для перехвата таких звонков. Номер, на который пойдет вызов, хранится в дополнительном параметре с ключом EXTRA_PHONE_NUMBER, а параметр resultData, содержащийся в возвращенном Намерении, указывает на номер, который действительно был вызван. Чтобы зарегистрировать Широковещательный приемник для этого действия, приложение должно содержать тег `uses-permission` с полномочием PROCESS_OUTGOING_CALL.
- ACTION_SCREEN_OFF и ACTION_SCREEN_ON. Передаются при выключении и включении экрана соответственно.
- ACTION_TIMEZONE_CHANGED. Это действие транслируется каждый раз, когда изменяется текущая временная зона в телефоне. Намерение содержит дополнительный параметр time-zone, в котором хранится идентификатор новой зоны из класса java.util.TimeZone.

Полный список транслируемых действий, используемых и передаваемых в Android, чтобы уведомлять приложения об изменениях состояния системы, находится по адресу http://developer.android.com/reference/android/content/Intent.html.

Android также использует Широковещательные намерения для оповещения о прикладных событиях, таких как получение входящих SMS-сообщений. Действия и Намерения, связанные с этими событиями, более подробно рассмотрены в следующих главах, из них вы узнаете больше о соответствующих Сервисах.

Знакомство с Ожидающими намерениями

Класс PendingIntent предоставляет механизм для создания Намерений, которые могут быть запущены сторонними приложениями спустя некоторое время.

Объекты PendingIntent, как правило, используются для упаковки таких Намерений, что должны сработать в ответ на будущее событие, такое как нажатие виджета или выбор элемента из панели уведомлений.

ПРИМЕЧАНИЕ

Ожидающие намерения (Pending Intents) запускают упакованные объекты Intent с теми же полномочиями, которые имеют сами, как будто вы делаете это из собственного приложения.

Как показано в листинге 5.22, класс PendingIntent предоставляет статические методы для создания Ожидающих намерений, которые используются для трансляции объектов Intent, а также для запуска Активностей или Сервисов.

Листинг 5.22. Создание новых Ожидающих намерений

```
// Запустите Активность
Intent startActivityIntent = new Intent(this, MyOtherActivity.class);
PendingIntent.getActivity(this, 0, startActivityIntent, 0);

// Транслируйте Намерение
Intent broadcastIntent = new Intent(NEW_LIFEFORM_DETECTED);
PendingIntent.getBroadcast(this, 0, broadcastIntent, 0);
```

Больше об Ожидающих намерениях вы узнаете в следующих главах, где они применяются для поддержки работы других Сервисов, таких как виджеты или уведомления.

Знакомство с Адаптерами

Адаптеры — связующее звено между классами, предоставляющими данные, и Представлениями (например, ListView), которые применяются в пользовательском интерфейсе. Адаптер отвечает за создание дочерних Представлений, отображающих каждый элемент внутри родительского виджета, и обеспечивают доступ к исходным данным.

Представления, поддерживающие привязку к Адаптеру, должны наследовать абстрактный класс AdapterView. Вы также можете создать собственный элемент управления на основе AdapterView и разработать новый класс Адаптера, чтобы обеспечить привязку.

Знакомство с некоторыми стандартными Адаптерами

Во многих случаях не придется создавать собственные Адаптеры с нуля. Android содержит набор Адаптеров, которые доставляют данные в стандартные элементы управления.

Поскольку Адаптеры отвечают и за доставку данных, и за создание Представлений для их отображения, они могут сильно повлиять на внешний вид и функциональность элемента, к которому привязаны.

Назовем два наиболее полезных и универсальных стандартных Адаптера.

- ArrayAdapter. Этот Адаптер использует механизм обобщенных типов для привязки AdapterView к массиву объектов определенного класса. По умолчанию ArrayAdapter задействует метод toString для каждого объекта в массиве, чтобы создать и наполнить данными элементы TextView. Альтернативные конструкторы позволяют применять более сложную разметку. Вы также можете наследовать этот класс, чтобы заменить экземпляры класса TextView на другие элементы (об этом в следующем разделе).
- SimpleCursorAdapter. Этот Адаптер прикрепляет Представление, заданное внутри разметки, к столбцам Курсора, ставшего результатом запроса к Источнику данных. Сперва нужно описать разметку в формате XML, а затем привязать каждый столбец к Представлению из этой разметки. Адаптер воплотит Представления для каждой записи из Курсора и наполнит их данными из соответствующего столбца.

В следующих разделах классы Адаптеров рассмотрены более подробно. В приведенных примерах демонстрируется привязывание данных к элементам ListView, но тот же принцип применим и к другим наследникам AdapterView, таким как Spinner и Gallery.

Настройка ArrayAdapter

По умолчанию ArrayAdapter использует метод toString из объекта массива, чтобы наполнять данными элемент TextView, размещенный внутри указанной разметки.

В большинстве случаев нужно будет изменять разметку, применяемую для отображения каждого Представления. Сделать это можно с помощью наследования класса ArrayAdapter, указывая конкретный тип и переопределяя

метод getView, чтобы установить свойства объекта для Представлений из разметки, как показано в листинге 5.23.

Метод getView используется для создания и наполнения данными Представления, отображаемого внутри родительского элемента AdapterView (например, ListView), который был привязан к исходному массиву с помощью этого Адаптера.

Метод getView принимает следующие параметры: позицию элемента, на какой он будет выведен, Представление, которое обновляется (или null), а также объект ViewGroup, где новое Представление поместится. Вызов метода getItem вернет значение из исходного массива по указанному индексу.

В результате метод getView должен вернуть экземпляр Представления, наполненный данными.

Листинг 5.23. Настройка ArrayAdapter

```
public class MyArrayAdapter extends ArrayAdapter<MyClass> {

  int resource;

  public MyArrayAdapter(Context context,
                        int _resource,
                        List<MyClass> items) {
    super(context, _resource, items);
    resource = _resource;
  }

  @Override
  public View getView(int position, View convertView, ViewGroup parent) {
    LinearLayout newView;

    MyClass classInstance = getItem(position);

    // TODO Извлечь из переменной classInstance
    // данные для отображения.

    // Inflate a new view if this is not an update.
    if (convertView == null) {
      newView = new LinearLayout(getContext());
      String inflater = Context.LAYOUT_INFLATER_SERVICE;
      LayoutInflater vi = (LayoutInflater)getContext().
getSystemService(inflater);
      vi.inflate(resource, newView, true);
    } else {
      newView = (LinearLayout)convertView;
    }

    // TODO Извлечь Представления для наполнения их данными.
    // TODO Наполнить Представления свойствами из полученного объекта.

    return newView;
  }
}
```

Использование Адаптеров для привязки данных

Чтобы применить Адаптер к классу, основанному на AdapterView, необходимо вызвать из Представления метод setAdapter, как показано в листинге 5.24.

Листинг 5.24. Создание и применение Адаптера

```
ArrayList<String> myStringArray = new ArrayList<String>();
ArrayAdapter<String> myAdapterInstance;

int layoutID = android.R.layout.simple_list_item_1;
myAdapterInstance = new ArrayAdapter<String>(this, layoutID ,
myStringArray);

myListView.setAdapter(myAdapterInstance);
```

В этом фрагменте представлен наиболее простой случай, когда привязка идет к массиву, состоящему из строк, и каждый элемент ListView являет собой одиночный объект TextView.

В первом из последующих примеров показано, как привязать массив сложных объектов к элементу ListView, использующему нестандартную разметку. Во втором продемонстрировано использование Адаптера SimpleCursorAdapter для привязки результатов запроса к нестандартной разметке внутри ListView.

Настройка Адаптера на примере проекта To-Do List

Данный пример[1] расширяет возможности проекта To-Do List, позволяя хранить все элементы списка в виде объектов ToDoItem, содержащих данные для каждого из них.

Нужно расширить класс ArrayAdapter, чтобы привязать набор объектов ToDoItem к элементу ListView, и изменить разметку, используемую для отображения каждого элемента списка.

1. Вернемся к проекту To-Do List. Создайте новый класс ToDoItem, который хранит задачу и время ее создания. Переопределите метод toString, чтобы возвращать совокупность данных, содержащихся в элементе.

```
package com.paad.todolist;

import java.text.SimpleDateFormat;
import java.util.Date;

public class ToDoItem {

  String task;
```

[1] Все фрагменты кода в этом примере — часть проекта Todo List из главы 5, их можно загрузить с сайта Wrox.com.

```
  Date created;

  public String getTask() {
    return task;
  }

  public Date getCreated() {
    return created;
  }

  public ToDoItem(String _task) {
    this(_task, new Date(java.lang.System.currentTimeMillis()));
  }

  public ToDoItem(String _task, Date _created) {
    task = _task;
    created = _created;
  }

  @Override
  public String toString() {
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yy");
    String dateString = sdf.format(created);
    return "(" + dateString + ") " + task;
  }
}
```

2. Откройте Активность ToDoList и измените типы переменных для ArrayList и ArrayAdapter, чтобы они могли хранить объекты ToDoItem вместо строк. Далее необходимо отредактировать метод onCreate, изменив инициализацию соответствующих полей. Нужно также обновить обработчик onKeyListener, добавив в него поддержку объектов ToDoItem.

```
private ArrayList<ToDoItem> todoItems;
private ListView myListView;
private EditText myEditText;
private ArrayAdapter<ToDoItem> aa;

@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  // Получите ссылку на ваше Представление
  setContentView(R.layout.main);

  // Получите ссылки на элементы UI
  myListView = (ListView)findViewById(R.id.myListView);
  myEditText = (EditText)findViewById(R.id.myEditText);

  todoItems = new ArrayList<ToDoItem>();
  int resID = R.layout.todolist_item;
  aa = new ArrayAdapter<ToDoItem>(this, resID, todoItems);
  myListView.setAdapter(aa);

  myEditText.setOnKeyListener(new OnKeyListener() {
```

```
    public boolean onKey(View v, int keyCode, KeyEvent event) {
      if (event.getAction() == KeyEvent.ACTION_DOWN)
        if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
          ToDoItem newItem = new ToDoItem(myEditText.getText().
toString());
          todoItems.add(0, newItem);
          myEditText.setText("");
          aa.notifyDataSetChanged();
          cancelAdd();
          return true;
        }
      return false;
    }
  });

  registerForContextMenu(myListView);
}
```

3. Теперь, если вы запустите Активность, она отобразит каждую задачу так, как показано на рис. 5.3.

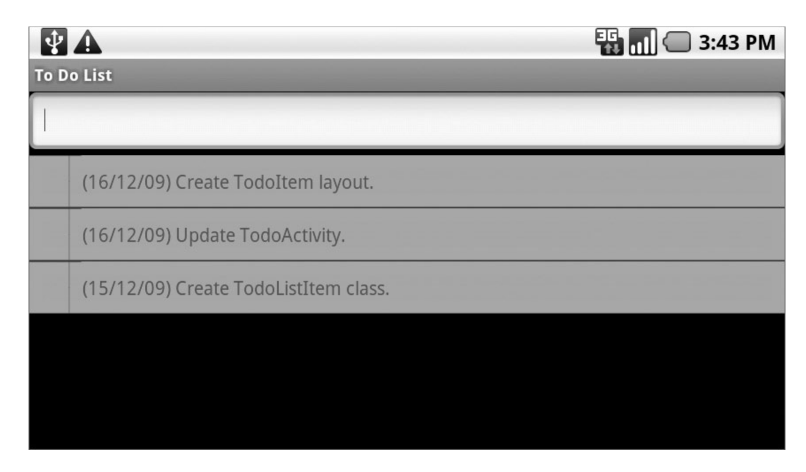


Рис. 5.3.

4. Пришло время создать нестандартную разметку для отображения каждой задачи.

   Начните с редактирования разметки, которую воплотили в главе 4, добавив в нее второй элемент TextView. Он будет использоваться для вывода даты создания каждого элемента.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="@color/notepad_paper">
  <TextView
    android:id="@+id/rowDate"
```

```
    android:layout_width="wrap_content" android:layout_height="fill_
parent"
    android:padding="10dp"
    android:scrollbars="vertical" android:fadingEdge="vertical"
    android:textColor="@color/notepad_text"
    android:layout_alignParentRight="true"
  />
  <TextView
    android:id="@+id/row"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:scrollbars="vertical" android:fadingEdge="vertical"
    android:textColor="@color/notepad_text"
    android:layout_alignParentLeft="@+id/rowDate"
  />
</RelativeLayout>
```

5. Создайте новый класс ToDoItemAdapter, который наследует ArrayAdapter и работает с экземплярами класса ToDoItem. Переопределите метод getView, чтобы передавать свойства task и created из объектов ToDoItem в Представления внутри разметки, которую вы создали в предыдущем пункте.

```
import java.text.SimpleDateFormat;
import android.content.Context;
import java.util.*;
import android.view.*;
import android.widget.*;

public class ToDoItemAdapter extends ArrayAdapter<ToDoItem> {

  int resource;

  public ToDoItemAdapter(Context _context,
                         int _resource,
                         List<ToDoItem> _items) {
    super(_context, _resource, _items);
    resource = _resource;
  }

  @Override
  public View getView(int position, View convertView, ViewGroup parent) {
    LinearLayout todoView;

    ToDoItem item = getItem(position);

    String taskString = item.getTask();
    Date createdDate = item.getCreated();
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yy");
    String dateString = sdf.format(createdDate);

    if (convertView == null) {
      todoView = new LinearLayout(getContext());
      String inflater = Context.LAYOUT_INFLATER_SERVICE;
```

```
      LayoutInflater vi = (LayoutInflater)getContext().
getSystemService(inflater);
      vi.inflate(resource, todoView, true);
    } else {
      todoView = (LinearLayout) convertView;
    }

    TextView dateView = (TextView)todoView.findViewById(R.id.rowDate);
    TextView taskView = (TextView)todoView.findViewById(R.id.row);

    dateView.setText(dateString);
    taskView.setText(taskString);

    return todoView;
  }
}
```

6. В завершение необходимо заменить объявление ArrayAdapter на ToDoItemAdapter.

```
private ToDoItemAdapter aa;
```

Внутри метода onCreate следует заменить создание экземпляра ArrayAdapter<String> на ToDoItemAdapter.

```
aa = new ToDoItemAdapter(this, resID, todoItems);
```

7. Если вы запустите свою Активность, она должна выглядеть, как на рис 5.4.

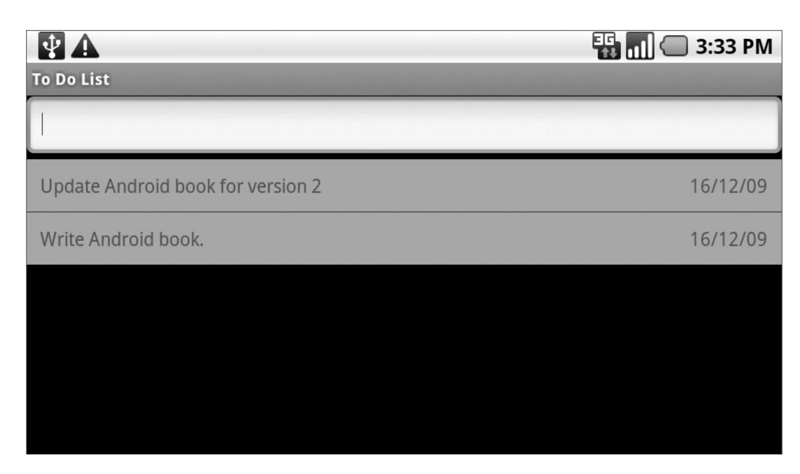


Рис. 5.4.

Использование SimpleCursorAdapter

SimpleCursorAdapter позволяет привязать Курсор к элементу ListView, используя собственное описание разметки, чтобы определить способ

отображения для каждой строки/элемента, которые заполняются данными из соответствующих столбцов.

Создайте экземпляр SimpleCursorAdapter, передав в его конструктор текущий контекст, ресурс с разметкой, Курсор и два массива: первый будет содержать названия используемых столбцов, второй (с той же размерностью) — идентификаторы ресурсов Представлений, применяемых для отображения содержимого соответствующих столбцов.

В листинге 5.25 показано, как создать SimpleCursorAdapter для отображения контактной информации.

Листинг 5.25. Создание SimpleCursorAdapter

```
String uriString = "content://contacts/people/";
Cursor myCursor = managedQuery(Uri.parse(uriString), null, null, null);

String[] fromColumns = new String[] {People.NUMBER, People.NAME};

int[] toLayoutIDs = new int[] { R.id.nameTextView, R.id.numberTextView};

SimpleCursorAdapter myAdapter;
myAdapter = new SimpleCursorAdapter(this,
                                    R.layout.simplecursorlayout,
                                    myCursor,
                                    fromColumns,
                                    toLayoutIDs);

myListView.setAdapter(myAdapter);
```

Ранее в этой главе SimpleCursorAdapter использовался в примере Contact Picker. В главе 7 вы подробнее познакомитесь с Источниками данных и Курсорами, а также найдете больше примеров применения SimpleCursorAdapter.

Ресурсы Интернета

Имея доступ к Интернету и браузеру, основанному на WebKit, вы можете спросить, существует ли хоть одна причина, по которой вместо веб-приложений нужно создавать родные аппликации, взаимодействующие с Глобальной сетью.

Есть немало преимуществ, которые вы получаете при создании родных тонких (thin) и толстых (rich) клиентов, вместо того, чтобы полностью полагаться на веб-технологии.

- **Пропускная способность.** Использование статических ресурсов, таких как изображения, HTML-страницы и аудиоданные, может оказаться довольно расточительным в условиях ограниченного и часто дорогого подключения к Сети. Создавая родные приложения, можно потреблять только нужные данные.

- **Кэширование.** Мобильный интернет-доступ пока еще не стал повсеместным. В сочетании с нестабильным сетевым подключением приложения, работающие в браузере, время от времени могут быть недоступны. Родное приложение способно кэшировать данные, чтобы предоставлять как можно больше функций без соединения с Интернетом.
- **Интеграция с системой.** Устройства на базе Android — нечто большее, чем просто платформа для запуска браузера: они могут включать геолокационные сервисы, уведомления, виджеты, камеры и акселерометры. Создавая родные приложения, вы можете сочетать данные, полученные из Сети, с аппаратными возможностями, доступными на устройстве, обеспечивая более тесное взаимодействие с пользователем.

Современные мобильные устройства предоставляют несколько альтернативных способов подключения к Интернету. В целом Android поддерживает два типа соединения, каждое из которых прозрачно на прикладном уровне:

- **мобильный Интернет** — доступ через GPRS, EDGE и 3G предоставляется мобильными операторами за плату;
- **Wi-Fi** — беспроводные адаптеры и мобильные точки доступа становятся все более распространенными.

Подключение к интернет-ресурсу

Хотя в этой книге и не рассматриваются подробности работы с конкретными веб-сервисами, вам не помешает познакомиться с общими принципами подключения к Интернету и получения входящего потока из удаленного источника данных.

Прежде чем вы сможете получить доступ к интернет-ресурсу, необходимо добавить в манифест своего приложения полномочие INTERNET, как показано в следующем фрагменте XML-кода:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

В листинге 5.26 демонстрируется простой шаблон для открытия потоков данных в Интернете.

Листинг 5.26. Открытие потока данных

```
String myFeed = getString(R.string.my_feed);
try {
  URL url = new URL(myFeed);

  URLConnection connection = url.openConnection();
```

```
    HttpURLConnection httpConnection = (HttpURLConnection)connection;

    int responseCode = httpConnection.getResponseCode();
    if (responseCode == HttpURLConnection.HTTP_OK) {
      InputStream in = httpConnection.getInputStream();
      [ ... В случае необходимости, обработайте входящей поток ... ]
    }
  }
  catch (MalformedURLException e) { }
  catch (IOException e) { }
```

Android содержит несколько классов, которые помогают управлять сетевыми соединениями. Они доступны в пакетах java.net.* и android.net.*.

Далее в этой главе представлен пример получения и обработки потоков из Интернета, содержащих список землетрясений за последние 24 часа.

В главе 13 содержится больше информации об управлении конкретными интернет-подключениями, в том числе об отслеживании состояния соединений и настройке подключения к беспроводным точкам доступа (Wi-Fi).

Использование интернет-ресурсов

Android предоставляет несколько способов использования интернет-ресурсов.

С одной стороны, вы можете задействовать объект WebView, чтобы встроить в свою Активность браузер, основанный на движке WebKit. С другой — есть возможность напрямую взаимодействовать с серверными процессами с помощью таких клиентских программных интерфейсов, как GData API от Google. Компромисс — использование удаленных потоков в формате XML, чтобы извлекать и обрабатывать данные с помощью стандартных обработчиков, поставляемых с Java, таких как SAX или его более эффективный аналог XmlPullParser.

Подробные инструкции о том, как обрабатывать XML и взаимодействовать с конкретными веб-сервисами, выходят за рамки тем этой книгой. Тем не менее в проекте Earthquake, с которым вы ознакомитесь далее в этой главе, представлен полностью рабочий пример разбора потоков в формате XML с помощью парсера SAX.

Используя интернет-ресурсы в своем приложении, помните, что сетевое подключение пользователя зависит от доступных ему коммуникационных технологий. Соединения через EDGE и GSM, как известно, обладают низкой пропускной способностью, тогда как подключение к Wi-Fi может отличаться ненадежностью, если его использовать в мобильном телефоне.

Оптимизируйте работу своего приложения, ограничив количество передаваемых им данных, и убедитесь, что оно способно справиться с нестабильным сетевым подключением и ограниченной пропускной способностью.

Знакомство с диалоговыми окнами

Диалоговые окна — неотъемлемая часть пользовательского интерфейса настольных, мобильных и веб-приложений. Они помогают пользователю отвечать на вопросы, выбирать и подтверждать действия. Кроме того, они отображают предупреждения и сообщения об ошибках. Диалоговые окна в Android — полупрозрачные «плавающие» Активности, частично перекрывающие родительский экран, из которого их вызвали.

Как показано на рис. 5.5, они чаще всего затеняют родительскую Активность позади себя с помощью фильтров размывания или затемнения.



Рис. 5.5.

Существует три способа реализации диалоговых окон в Android.

- **Использование класса Dialog (или его производных).** Наряду с универсальным классом AlertDialog Android содержит специальные диалоговые окна, наследованные от Dialog. Каждое из них создано для предоставления специфических диалоговых функций. Экран, использующий класс Dialog, создается внутри родительской Активности и находится под ее полным контролем, поэтому нет нужды регистрировать его в манифесте.
- **Активности, использующие визуальный стиль диалоговых окон.** Вы можете применить визуальный стиль для диалогов к обычной Активности, чтобы она выглядела, как стандартное диалоговое окно.

- **Уведомления типа Toast.** Объекты Toast — специальные немодальные кратковременные окна с сообщениями, часто используемые Приемниками широковещательных намерений и Сервисами, чтобы оповещать пользователей о событиях, происходящих в фоне. Более подробно с уведомлениями типа Toast вы сможете познакомиться в главе 9.

Знакомство с классами диалоговых окон

Чтобы использовать базовый класс Dialog, необходимо создать новый экземпляр и установить для него заголовок и разметку, используя методы setTitle и setContentView, как показано в листинге 5.27.

Листинг 5.27. Создание нового диалогового окна с помощью класса Dialog

```
Dialog d = new Dialog(MyActivity.this);

// Новое окно затеняет и размывает Активность,
// которую оно перекрывает.
Window window = d.getWindow();
window.setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
                WindowManager.LayoutParams.FLAG_BLUR_BEHIND);

// Установите заголовок
d.setTitle("Dialog Title");
// Передайте ссылку на разметку
d.setContentView(R.layout.dialog_view);

// Найдите элемент TextView внутри вашей разметки
// и установите ему соответствующий текст
TextView text = (TextView)d.findViewById(R.id.dialogTextView);
text.setText("This is the text in my dialog");
```

Настроив диалоговое окно надлежащим образом, вызовите метод show, чтобы отобразить его на экране:

```
d.show();
```

Класс AlertDialog

AlertDialog — одна из наиболее универсальных реализаций класса Dialog. Она предоставляет параметры, позволяющие создавать диалоговые окна для использования в некоторых часто встречающихся ситуациях.

- Вывод сообщения и предложение пользователю выбрать один из трех вариантов в виде кнопок. Скорее всего, вы уже хорошо знакомы с подобного рода функциональностью, если когда-нибудь занимались программированием под настольные платформы, где вышеупомянутый выбор, как правило, представлен сочетанием кнопок OK, Cancel, Yes и No.
- Предложение списка вариантов в виде флажков (CheckBox) или переключателей (RadioButton).

- Предоставление поля ввода, куда пользователь может записать текстовые данные.

Чтобы построить пользовательский интерфейс для AlertDialog, создайте новый объект AlertDialog.Builder:

```
AlertDialog.Builder ad = new AlertDialog.Builder(context);
```

После этого можете установить значения для заголовка и сообщения, которое необходимо отобразить, а также, если потребуется, для кнопок, флажков/переключателей и полей ввода, которые хотите показать. Вы также можете задать обработчики для событий, чтобы взаимодействовать с пользователем.

В листинге 5.28 представлен пример использования AlertDialog для отображения сообщения, когда пользователю предлагается нажать одну из двух кнопок на выбор. Оба варианта приводят к тому, что выполняется прикрепленный обработчик OnClickListener и закрывается диалоговое окно.

Листинг 5.28. Настройка AlertDialog

```
Context context = MyActivity.this;
String title = "It is Pitch Black";
String message = "You are likely to be eaten by a grue.";
String button1String = "Go Back";
String button2String = "Move Forward";

AlertDialog.Builder ad = new AlertDialog.Builder(context);
ad.setTitle(title);
ad.setMessage(message);
ad.setPositiveButton(button1String,

                     new OnClickListener() {
                       public void onClick(DialogInterface dialog,
int arg1) {
                         eatenByGrue();
                       }
                     });
ad.setNegativeButton(button2String,
                     new OnClickListener(){
                       public void onClick(DialogInterface dialog,
int arg1) {
                         // заглушка
                       }
                     });
ad.setCancelable(true);
ad.setOnCancelListener(new OnCancelListener() {
                             public void onCancel(DialogInterface
dialog) {
                               eatenByGrue();
                             }
                       });
```

Чтобы отобразить созданный вами диалог, вызовите метод show:

```
ad.show();
```

Рекомендуется использовать обработчики Активности onCreateDialog и onPrepareDialog, чтобы создавать экземпляры диалогов, которые способны сохранять свое состояние. Данный подход будет рассмотрен далее в этой главе.

Специальные диалоговые окна для ввода данных

Один из наиболее частых способов использования диалоговых окон — предоставление интерфейса для пользовательского ввода. Android содержит несколько специальных диалоговых окон, включающих элементы управления, созданные для обеспечения ввода данных пользователем. Вот некоторые из них.

- CharacterPickerDialog. Позволяет пользователю выбрать символ с диакритическим знаком (ударением), основанный на обычном исходном символе.
- DatePickerDialog. Позволяет выбрать дату с помощью элемента DatePicker. Конструктор содержит функцию обратного вызова, которая уведомляет родительскую Активность об установленной дате.
- TimePickerDialog. Аналогично DatePickerDialog дает возможность выбрать время с помощью элемента TimePicker.
- ProgressDialog. Диалоговое окно, отображающее элемент ProgressBar и текстовое сообщение под ним. Идеально подходит для информирования пользователя о ходе выполнения длительных операций.

Использование Активностей в качестве диалоговых окон

Диалоговые окна предоставляют простой и легковесный способ отображения информации, но иногда вам нужно больше контроля над жизненным циклом и содержимым диалогов.

Решение — реализация диалогового окна в виде полноценной Активности. Создавая Активность, вы теряете легковесность класса Dialog, но взамен получаете возможность реализовать любой экран на свой вкус. При этом появляется доступ к обработчикам событий жизненного цикла самой Активности.

Самый простой способ сделать Активность похожей на диалоговое окно — применить соответствующий визуальный стиль. Для этого необходимо

добавить атрибут android:style/Theme.Dialog в тег Активности в манифесте, как показано в следующем фрагменте:

```
<activity android:name="MyDialogActivity"
          android:theme="@android:style/Theme.Dialog">
</activity>
```

Это заставит вашу Активность вести себя, как диалоговое окно — она будет находиться в «плавающем» состоянии поверх родительского компонента, частично его перекрывая.

Отображение диалоговых окон и управление ими

Вместо того чтобы создавать новые экземпляры диалоговых окон каждый раз, когда это необходимо, можно использовать обработчики событий onCreateDialog и onPrepareDialog, предоставляемые Android. Вызывая их внутри класса Активности, можно управлять диалоговыми окнами, сохраняя их экземпляры.

Переопределив обработчик onCreateDialog, укажите конкретные диалоги, которые будут созданы при вызове метода showDialog. Как показано в листинге 5.29, переопределенный метод содержит оператор switch, позволяющий определить, какое именно диалоговое окно следует отобразить.

Листинг 5.29. Использование обработчика onCreateDialog

```
static final private int TIME_DIALOG = 1;

@Override
public Dialog onCreateDialog(int id) {
  switch(id) {
    case (TIME_DIALOG) :
      AlertDialog.Builder timeDialog = new AlertDialog.Builder(this);
      timeDialog.setTitle("The Current Time Is...");
      timeDialog.setMessage("Now");
      return timeDialog.create();
  }
  return null;
}
```

После начального создания при каждом вызове метода showDialog будет срабатывать обработчик onPrepareDialog. Переопределив этот метод, вы можете изменять диалоговое окно при каждом его выводе на экран. Это позволит привнести контекст в любое из отображаемых значений. В листинге 5.30 диалог, созданный выше, выводит сообщение о текущем времени.

Листинг 5.30. Использование обработчика onPrepareDialog

```
@Override
public void onPrepareDialog(int id, Dialog dialog) {
  switch(id) {
    case (TIME_DIALOG) :
```

```
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        Date currentTime = new Date(java.lang.System.currentTimeMillis());
        String dateString = sdf.format(currentTime);
        AlertDialog timeDialog = (AlertDialog)dialog;
        timeDialog.setMessage(dateString);

        break;
    }
}
```

Переопределив эти обработчики, вы можете отображать диалоги с помощью вызова метода showDialog, как показано ниже. Передайте в качестве параметра идентификатор диалога, который вы хотите вывести на экран, и Android сам создаст его (если нужно) и подготовит к отображению:

```
showDialog(TIME_DIALOG);
```

Кроме улучшенного механизма управления ресурсами этот подход позволяет вашей Активности контролировать целостность информации о состоянии диалогов. Любой выбор элементов или набор текста в поле ввода сохранятся для каждого экземпляра диалогового окна независимо от того, сколько раз они будут отображаться на экране.

Создание приложения Earthquake Viewer

В следующем примере[1] вы создадите инструмент, который использует информационный поток Геологической Службы США (USGS) для отображения списка последних землетрясений.

ПРИМЕЧАНИЕ

Позже вы неоднократно будете возвращаться к этому примеру. В главах 6 и 7 добавите сохранение настроек и распространение информации о землетрясениях через Источник данных, в главах 8 и 9 внедрите поддержку картографии и переместите процесс обновления исходных данных в Сервис.

В этом примере вы создадите Активность, основанную на ListView, которая подключается к потоку данных и выводит на экран местоположение, магнитуду и время начала содержащихся там землетрясений. Используйте объект AlertDialog для отображения окна, которое содержит элемент TextView со ссылкой на сайт организации USGS.

1. Начните с создания проекта Earthquake, содержащего одноименную Активность. Отредактируйте ресурс с разметкой `main.xml`, добавив в него элемент ListView (не забудьте задать соответствующий идентификатор,

[1] Все фрагменты кода в этом примере — часть проекта Earthquake из главы 5, их можно загрузить с сайта Wrox.com.

чтобы потом можно было получить на него ссылку внутри кода Активности).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <ListView
    android:id="@+id/earthquakeListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
</LinearLayout>
```

2. Создайте новый публичный класс Quake. Он нужен для хранения информации (дата, подробности, местоположение, магнитуда и ссылка) о каждом землетрясении. Переопределите метод toString, чтобы предоставить строку, с помощью которой каждое землетрясение будет представлено в списке ListView.

```
package com.paad.earthquake;

import java.util.Date;
import java.text.SimpleDateFormat;
import android.location.Location;

public class Quake {
  private Date date;
  private String details;
  private Location location;
  private double magnitude;
  private String link;

  public Date getDate() { return date; }
  public String getDetails() { return details; }
  public Location getLocation() { return location; }
  public double getMagnitude() { return magnitude; }
  public String getLink() { return link; }

  public Quake(Date _d, String _det, Location _loc, double _mag, String
_link) {
    date = _d;
    details = _det;
    location = _loc;
    magnitude = _mag;
    link = _link;
  }

  @Override
  public String toString() {
    SimpleDateFormat sdf = new SimpleDateFormat("HH.mm");
```

```
    String dateString = sdf.format(date);
    return dateString + ": " + magnitude + " " + details;
  }

}
```

3. Внутри Активности Earthquake переопределите метод onCreate, чтобы сохранить массив объектов Quake и привязать его к элементу ListView с помощью ArrayAdapter.

```
package com.paad.earthquake;

import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.GregorianCalendar;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import android.app.Activity;
import android.app.Dialog;
import android.location.Location;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.WindowManager;
import android.view.MenuItem;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class Earthquake extends Activity {

  ListView earthquakeListView;
  ArrayAdapter<Quake> aa;

  ArrayList<Quake> earthquakes = new ArrayList<Quake>();

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
```

```
    setContentView(R.layout.main);

    earthquakeListView = (ListView)this.findViewById(R.
id.earthquakeListView);

    int layoutID = android.R.layout.simple_list_item_1;
    aa = new ArrayAdapter<Quake>(this, layoutID , earthquakes);
    earthquakeListView.setAdapter(aa);
  }
}
```

4. Начните обработку потока. В этом примере используется суточная лента землетрясений, имеющих магнитуду больше 2,5.

СОВЕТ

Добавьте адрес ленты в виде внешнего строкового ресурса, что позволит заменить ее на другую, указанную пользователем.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Earthquake</string>
  <string name="quake_feed">
    http://earthquake.usgs.gov/eqcenter/catalogs/1day-M2.5.xml
  </string>
</resources>
```

5. Прежде чем приложение получит доступ к Интернету, ему должно быть присвоено соответствующее полномочие. Добавьте тег `uses-permission` в манифест.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

6. Снова вернитесь к Активности Earthquake и создайте новый метод refreshEarthquakes, который будет подключаться к потоку землетрясений и обрабатывать его. Извлеките каждое землетрясение и сопутствующие данные, чтобы получить дату, магнитуду, ссылку и местоположение. Завершая обработку нового землетрясения, передавайте его в метод addNewQuake.

ПРИМЕЧАНИЕ

В данном примере лента землетрясений в формате XML разбирается с помощью парсера SAX. Тем не менее существует несколько других парсеров, включая XmlPullParser. Анализ всех альтернативных способов разбора данных в формате XML (и их использование) выходит за рамки этой книги, однако важно адекватно оценивать и сравнивать различные варианты, доступные для ваших собственных приложений.

```
private void refreshEarthquakes() {
  // Получите XML
  URL url;
```

```
  try {
    String quakeFeed = getString(R.string.quake_feed);
    url = new URL(quakeFeed);

    URLConnection connection;
    connection = url.openConnection();

    HttpURLConnection httpConnection = (HttpURLConnection)connection;
    int responseCode = httpConnection.getResponseCode();

    if (responseCode == HttpURLConnection.HTTP_OK) {
      InputStream in = httpConnection.getInputStream();

      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();

      // Разберите поток землетрясений.
      Document dom = db.parse(in);
      Element docEle = dom.getDocumentElement();

      // Очистите старые землетрясения
      earthquakes.clear();

      // Получите список всех землетрясений.
      NodeList nl = docEle.getElementsByTagName("entry");
      if (nl != null && nl.getLength() > 0) {
        for (int i = 0 ; i < nl.getLength(); i++) {
          Element entry = (Element)nl.item(i);
          Element title = (Element)entry.getElementsByTagName("title").
item(0);
          Element g = (Element)entry.getElementsByTagName("georss:point").
item(0);
          Element when = (Element)entry.getElementsByTagName("updated").
item(0);
          Element link = (Element)entry.getElementsByTagName("link").
item(0);

          String details = title.getFirstChild().getNodeValue();
          String hostname = "http://earthquake.usgs.gov";
          String linkString = hostname + link.getAttribute("href");

          String point = g.getFirstChild().getNodeValue();
          String dt = when.getFirstChild().getNodeValue();
          SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-
dd'T'hh:mm:ss'Z'");
          Date qdate = new GregorianCalendar(0,0,0).getTime();
          try {
            qdate = sdf.parse(dt);
          } catch (ParseException e) {
            e.printStackTrace();
          }

          String[] location = point.split(" ");
          Location l = new Location("dummyGPS");
          l.setLatitude(Double.parseDouble(location[0]));
          l.setLongitude(Double.parseDouble(location[1]));

          String magnitudeString = details.split(" ")[1];
```

```
          int end = magnitudeString.length()-1;
          double magnitude = Double.parseDouble(magnitudeString.
substring(0, end));

          details = details.split(",")[1].trim();

          Quake quake = new Quake(qdate, details, l, magnitude,
linkString);

          // Обработайте только что найденное землетрясение
          addNewQuake(quake);
        }
      }
    }
  } catch (MalformedURLException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  } catch (ParserConfigurationException e) {
    e.printStackTrace();
  } catch (SAXException e) {
    e.printStackTrace();
  }
  finally {
  }
}

private void addNewQuake(Quake _quake) {
  // TODO: Добавьте землетрясения в список.
}
```

7. Измените метод addNewQuake таким образом, чтобы он мог принимать и добавлять каждое новое землетрясение в список ArrayList. Он также должен оповещать ArrayAdapter об изменениях в исходных данных.

```
private void addNewQuake(Quake _quake) {
  // Добавьте новое землетрясение в список.
  earthquakes.add(_quake);

  // Оповестите ArrayAdapter об изменениях.
  aa.notifyDataSetChanged();
}
```

8. Отредактируйте обработчик onCreate. Для этого добавьте вызов метода refreshEarthquakes при запуске Активности.

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  setContentView(R.layout.main);

  earthquakeListView = (ListView)this.findViewById(R.
id.earthquakeListView);

  int layoutID = android.R.layout.simple_list_item_1;
```

```
  aa = new ArrayAdapter<Quake>(this, layoutID , earthquakes);
  earthquakeListView.setAdapter(aa);

  refreshEarthquakes();
}
```

ПРИМЕЧАНИЕ

В нашем примере сетевые запросы осуществляются в главном графическом потоке. Это не лучший подход, так как приложение перестанет реагировать на события, если запрос занимает более нескольких секунд. В главе 9 вы научитесь переносить трудоемкие (требующие много времени на выполнение) операции вроде этой в фоновый Сервис.

9. Запустив свой проект, вы должны увидеть элемент ListView, содержащий землетрясения за последние 24 часа, у которых магнитуда превышает 2,5. Внешний вид приложения показан на рис. 5.6.

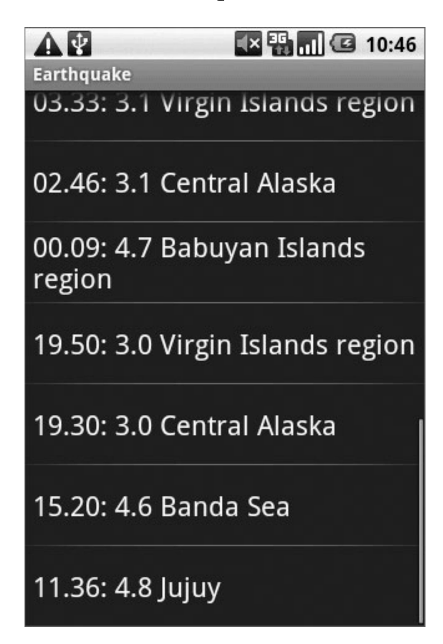


Рис. 5.6.

10. Необходимо выполнить еще два шага, чтобы сделать приложение более полезным. Прежде всего создайте новый пункт меню, чтобы пользователь при желании сам мог обновлять ленту землетрясений.

    10.1. Начните с добавления нового внешнего строкового ресурса для пункта меню.

```
<string name="menu_update">
 Refresh Earthquakes
</string>
```

10.2. Затем переопределите методы Активности onCreateOptionsMenu и onOptionsItemSelected, чтобы отображать пункт меню **Refresh Earthquakes** и обрабатывать его нажатие.

```
static final private int MENU_UPDATE = Menu.FIRST;

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  menu.add(0, MENU_UPDATE, Menu.NONE, R.string.menu_update);

  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  super.onOptionsItemSelected(item);

  switch (item.getItemId()) {
    case (MENU_UPDATE): {
      refreshEarthquakes();
      return true;
    }
  }
  return false;
}
```

11. Теперь добавьте немного интерактивности. Сделайте так, чтобы при выборе пользователем землетрясения из списка на экране выводилось соответствующее диалоговое окно.

11.1. Создайте новый ресурс с разметкой `quake_details.xml` для диалогового окна, которое будет отображаться при выборе элемента.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dp">
  <TextView
    android:id="@+id/quakeDetailsTextView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textSize="14sp"
  />
</LinearLayout>
```

11.2. Отредактируйте метод onCreate, добавив в него реализацию интерфейса ItemClickListener для ListView, чтобы выводить на экран диалоговое окно при нажатии элемента с землетрясением.

```
static final private int QUAKE_DIALOG = 1;
Quake selectedQuake;

@Override
```

```
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  setContentView(R.layout.main);

  earthquakeListView = (ListView)this.findViewById(R.
id.earthquakeListView);
  earthquakeListView.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView _av, View _v, int _index,
       long arg3) {
      selectedQuake = earthquakes.get(_index);
      showDialog(QUAKE_DIALOG);
    }
  });

  int layoutID = android.R.layout.simple_list_item_1;
  aa = new ArrayAdapter<Quake>(this, layoutID , earthquakes);
  earthquakeListView.setAdapter(aa);

  refreshEarthquakes();
}
```

11.3. Переопределите методы onCreateDialog и onPrepareDialog, чтобы создать и заполнить данными объект AlertDialog.

```
@Override
public Dialog onCreateDialog(int id) {
  switch(id) {
    case (QUAKE_DIALOG) :
      LayoutInflater li = LayoutInflater.from(this);
      View quakeDetailsView = li.inflate(R.layout.quake_details, null);

      AlertDialog.Builder quakeDialog = new AlertDialog.Builder(this);
      quakeDialog.setTitle("Quake Time");
      quakeDialog.setView(quakeDetailsView);
      return quakeDialog.create();
  }
  return null;
}

@Override
public void onPrepareDialog(int id, Dialog dialog) {
  switch(id) {
    case (QUAKE_DIALOG) :
      SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
      String dateString = sdf.format(selectedQuake.getDate());
      String quakeText = "Magnitude " + selectedQuake.getMagnitude() +
                         "\n" + selectedQuake.getDetails() + "\n" +
                         selectedQuake.getLink();

      AlertDialog quakeDialog = (AlertDialog)dialog;
      quakeDialog.setTitle(dateString);
      TextView tv = (TextView)quakeDialog.findViewById
         (R.id.quakeDetailsTextView);
      tv.setText(quakeText);

      break;
  }
}
```

11.4. В завершение примените к диалогу Linkify, чтобы создать гиперссылку на сайт организации USGS. Поправьте описание разметки диалогового окна, включив в нее атрибут `autolink`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dp">
  <TextView
    android:id="@+id/quakeDetailsTextView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textSize="14sp"
    android:autoLink="all"
  />
</LinearLayout>
```

Еще раз запустите свое приложение. Если щелкнуть на каком-нибудь землетрясении, появится диалоговое окно и частично перекроет список, как показано на рис. 5.7.



Рис. 5.7.

Резюме

Основное внимание в этой главе уделялось интеграции компонентов вашего приложения.

Намерения предоставляют универсальную систему обмена сообщениями, которая позволяет передавать данные как внутри одного приложения, так и между несколькими процессами, выполняя действия и реагируя на события. Вы научились использовать явные и неявные Намерения, чтобы запускать новые Активности, узнали, как динамически заполнять данными меню Активностей, используя Фильтры намерений.

Познакомились с Широковещательными намерениями и поняли, как с их помощью отправлять сообщения в контексте всей системы, в частности для поддержки событийной модели, основанной на системных и прикладных событиях.

Вы узнали, как использовать дочерние Активности для передачи данных между компонентами, как применять диалоговые окна для отображения информации и для упрощения пользовательского ввода.

Познакомились с Адаптерами, которые использовались для привязки исходных данных к визуальным компонентам. В частности, узнали, как применять ArrayAdapter и SimpleCursorAdapter для привязки элемента ListView к массивам ArrayList и Курсорам.

Наконец, вы усвоили основные принципы подключения к Интернету и применения удаленных потоков в качестве источников данных для родных клиентских приложений.

Вы также узнали:

- как использовать Linkify для добавления неявных намерений в элементы TextView во время выполнения программы;
- какие стандартные действия в Android доступны для расширения, замены или перехвата;
- как использовать Фильтры намерений, чтобы позволить собственным Активностям обрабатывать дополнительные действия, которые отправляются вашими или любыми другими приложениями;
- каким образом отслеживать транслируемые Намерения, используя Широковещательные приемники;
- как использовать Активности в качестве диалоговых окон.

В следующей главе вы научитесь обеспечивать постоянство данных внутри своих приложений. Android предоставляет механизмы для сохранения данных приложения, включая файлы, простые настройки и полноценные реляционные базы данных (используя библиотеку SQLite). В главе 6 основное внимание уделено использованию настроек и сохранению состояний Активности, в главе 7 рассмотрены Источники данных и базы данных на основе SQLite.

Глава 6
ФАЙЛЫ, СОХРАНЕНИЕ СОСТОЯНИЯ И НАСТРОЙКИ

Содержание главы

- Хранение простых данных приложения.
- Сохранение данных Активности между сессиями.
- Создание Экранов настроек (Preference Screens) и управление настройками приложения.
- Сохранение и загрузка файлов, работа с локальной файловой системой.
- Включение статических файлов в качестве внешних ресурсов.

В этой главе вы познакомитесь с двумя самыми простыми, но в то же время наиболее гибкими способами обеспечения постоянства данных в Android — Общими настройками (Shared Preferences) и локальными файлами. Для большинства приложений сохранение и загрузка данных играют важную роль. Как минимум Активность должна сохранять состояние пользовательского интерфейса (UI) каждый раз, когда она переходит в фоновый режим — так гарантируется такое же состояние UI, когда Активность опять возвращается на передний план, даже если перед этим процесс был убит и перезапущен.

Вполне возможно, что вам понадобится сохранять пользовательские настройки приложения, изменения в UI или введенные данные. В связи со свойственной Android неопределенностью времени жизни Активности и приложения особенно важно сохранение целостности состояния UI и данных приложения между сессиями. Android предоставляет несколько альтернативных способов сохранения данных приложения, каждый из которых оптимизирован для определенных нужд.

Общие настройки — простой и легкий механизм, основанный на парах «ключ — значение» и предназначенный для сохранения примитивных данных приложения, чаще всего пользовательских настроек. Android также обеспечивает доступ к локальной файловой системе через специальные методы и через обычные классы Java.IO.

Сохранение простых данных приложения

Методы обеспечения постоянства данных в Android дают возможность выбирать между скоростью, эффективностью и надежностью.

- **Общие настройки.** Легкий механизм для записи заранее известного набора значений при хранении состояния UI, пользовательских или программных настроек. Общие настройки позволяют сохранять группы пар «ключ — значение», содержащие примитивные данные в виде именных настроек.
- **Сохранение состояния приложения**. Активности содержат специальные обработчики событий, предназначенные для записи текущего состояния пользовательского интерфейса в момент, когда приложение переходит в фоновый режим.
- **Файлы**. Иногда запись в файл и чтение из него — единственный возможный вариант. Android позволяет создавать и загружать локальные файлы, используя внутренние или внешние носители, подключенные к устройству.

Есть два «легковесных» метода сохранения простых программных данных в приложениях для Android — Общие настройки и пара обработчиков событий, предназначенных для сохранения информации об экземпляре Активности. Оба механизма используют принцип пар «ключ — значение», чтобы хранить простые, примитивные значения.

Класс SharedPreferences позволяет создавать в приложении именованные ассоциативные массивы типа «ключ — значение», которые могут быть использованы различными компонентами приложения (работая при этом в контексте одного и того же приложения).

Общие настройки поддерживают базовые типы boolean, string, float, long и integer, что делает их идеальным средством для быстрого сохранения значений по умолчанию, переменных экземпляра класса, текущего состояния UI и пользовательских настроек. Они чаще всего используются для обеспечения постоянства данных между пользовательскими сессиями и доступа к ним компонентов приложения.

Активности также предоставляют обработчик onSaveInstanceState — специально для сохранения состояния пользовательского интерфейса, когда работа Активности может быть завершена из-за нехватки ресурсов.

Этот обработчик функционирует по тому же принципу, что и Общие настройки, и предлагает параметр Bundle — ассоциативный массив вида «ключ — значение», который поддерживает примитивные типы и может быть использован для сохранения переменных экземпляра Активности. Bundle — параметр, что передается в методы onCreate и onRestoreInstanceState. Bundle, содержащий состояние UI, должен быть использован для записи значений, необходимых

Активности, чтобы вывести на экран тот самый пользовательский интерфейс, который отображался до непредвиденного закрытия.

Создание и сохранение настроек

Чтобы создать или изменить Общие настройки, нужно вызвать метод getSharedPreferences в контексте приложения, передавая имя Общих настроек, которые вы хотите изменить. Общие настройки доступны для компонентов приложения, но не для других приложений.

Чтобы изменить Общие настройки, используйте класс SharedPreferences. Editor. Получите объект Editor, вызвав метод edit объекта SharedPreferences, который вы хотите изменить. Чтобы сохранить изменения, вызовите метод commit объекта Editor, как показано в листинге 6.1.

Листинг 6.1. Создание новых Общих настроек

```
// Получите доступ к объекту Editor, чтобы изменить общие настройки.
SharedPreferences.Editor editor = mySharedPreferences.edit();

// Задайте новые базовые типы в объекте общих настроек.
editor.putBoolean("isTrue", true);
editor.putFloat("lastFloat", 1f);
editor.putInt("wholeNumber", 2);
editor.putLong("aNumber", 3l);
editor.putString("textEntryValue", "Not Empty");

// Сохраните изменения.
editor.commit();
}
```

Получение Общих настроек

Как и в случае с изменением и сохранением, доступ к Общим настройкам обеспечивает метод getSharedPreferences. Передайте имя тех Общих настроек, доступ к которым хотите получить, и используйте типизированный метод get<тип>, чтобы извлечь сохраненные значения.Каждому геттеру передаются ключ и значение по умолчанию (используется в том случае, если для данного ключа пока что не сохранено никакое значение), как показано в листинге 6.2.

Листинг 6.2. Получение сохраненных Общих настроек

```
public static String MY_PREFS = "MY_PREFS";
public void loadPreferences() {
  // Получите объект настроек.
  int mode = Activity.MODE_PRIVATE;
  SharedPreferences mySharedPreferences = getSharedPreferences(MY_PREFS,
mode);

  // Получите сохраненные значения.
  boolean isTrue = mySharedPreferences.getBoolean("isTrue", false);
```

```
    float lastFloat = mySharedPreferences.getFloat("lastFloat", 0f);
    int wholeNumber = mySharedPreferences.getInt("wholeNumber", 1);
    long aNumber = mySharedPreferences.getLong("aNumber", 0);
    String stringPreference = mySharedPreferences.
getString("textEntryValue", "");
}
```

Создание Активности для настроек приложения Earthquake Viewer

В главе 5 вы создали приложение Earthquake Viewer, которое выводило список недавних землетрясений, основанный на ленте RSS.

В следующем примере[1] вы создадите Активность для настройки данного приложения. Это позволит пользователю менять настройки, подстраивая программу под себя, — включать и выключать автоматические обновления, контролировать их частоту и указывать минимальную магнитуду, при которой землетрясение будет выводиться на экран.

ПРИМЕЧАНИЕ

Позже в этой главе вы замените эту Активность на стандартный Экран настроек.

1. Откройте проект Earthquake, созданный в главе 5. Добавьте новые строковые ресурсы для тех меток, что будут отображаться в Экране настроек. Добавьте строку для нового пункта меню, с помощью которого пользователь сможет получить доступ к этой Активности:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Earthquake</string>
  <string name="quake_feed">
    http://earthquake.usgs.gov/eqcenter/catalogs/1day-M2.5.xml
  </string>
  <string name="menu_update">Refresh Earthquakes</string>
  <string name="auto_update_prompt">Auto Update?</string>
  <string name="update_freq_prompt">Update Frequency</string>
  <string name="min_quake_mag_prompt">Minimum Quake Magnitude</string>
  <string name="menu_preferences">Preferences</string>
</resources>
```

2. Создайте новый ресурс разметки `preferences.xml` для Активности Preferences. Добавьте CheckBox, чтобы показывать состояние переключателя «автоматическое обновление», а также элементы Spinner — для выбора частоты обновлений и фильтра магнитуд.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
```

[1] Все фрагменты кода в этом примере — часть проекта Earthquake из главы 6, их можно загрузить с сайта Wrox.com.

```
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/auto_update_prompt"
  />
  <CheckBox android:id="@+id/checkbox_auto_update"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/update_freq_prompt"
  />
  <Spinner android:id="@+id/spinner_update_freq"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:drawSelectorOnTop="true"
  />
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/min_quake_mag_prompt"
  />
  <Spinner android:id="@+id/spinner_quake_mag"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:drawSelectorOnTop="true"  />
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <Button android:id="@+id/okButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@android:string/ok"
    />
    <Button android:id="@+id/cancelButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@android:string/cancel"
    />
  </LinearLayout>
</LinearLayout>
```

3. Создайте четыре ресурса с массивами в новом файле `res/values/arrays.xml`. Значения, которые они содержат, будут использованы элементами Spinner для отображения частоты обновлений и минимальной магнитуды:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="update_freq_options">
    <item>Every Minute</item>
```

```
    <item>5 minutes</item>
    <item>10 minutes</item>
    <item>15 minutes</item>
    <item>Every Hour</item>
  </string-array>

  <array name="magnitude">
    <item>3</item>
    <item>5</item>
    <item>6</item>
    <item>7</item>
    <item>8</item>
  </array>

  <string-array name="magnitude_options">
    <item>3</item>
    <item>5</item>
    <item>6</item>
    <item>7</item>
    <item>8</item>
  </string-array>

  <array name="update_freq_values">
    <item>1</item>
    <item>5</item>
    <item>10</item>
    <item>15</item>
    <item>60</item>
  </array>
</resources>
```

4. Создайте Активность Preferences.

   Переопределите метод onCreate, чтобы дополнить разметку, созданную в пункте 2, и получите ссылки на CheckBox и оба виджета Spinner. Затем вызовите заглушку populateSpinners.

```
package com.paad.earthquake;
import android.app.Activity;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Spinner;

public class Preferences extends Activity {

  CheckBox autoUpdate;
  Spinner updateFreqSpinner;
  Spinner magnitudeSpinner;

  @Override
  public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.preferences);

    updateFreqSpinner = (Spinner)findViewById(R.id.spinner_update_freq);
    magnitudeSpinner = (Spinner)findViewById(R.id.spinner_quake_mag);
    autoUpdate = (CheckBox)findViewById(R.id.checkbox_auto_update);

    populateSpinners();
  }

  private void populateSpinners() {
  }
}
```

5. Заполните кодом метод populateSpinners, используя Адаптеры данных (ArrayAdapter) для привязки каждого элемента Spinner к соответствующему массиву:

```
private void populateSpinners() {
  // Заполните Spinner данными о частоте обновления
  ArrayAdapter<CharSequence> fAdapter;
  fAdapter = ArrayAdapter.createFromResource(this, R.array.update_freq_
                      options, android.R.layout.simple_spinner_item);
  int spinner_dd_item = android.R.layout.simple_spinner_dropdown_item;
  fAdapter.setDropDownViewResource(spinner_dd_item);
  updateFreqSpinner.setAdapter(fAdapter);
    // Заполните Spinner данными о минимальной магнитуде
    ArrayAdapter<CharSequence> mAdapter;
    mAdapter = ArrayAdapter.createFromResource(this,
      R.array.magnitude_options,
      android.R.layout.simple_spinner_item);
    mAdapter.setDropDownViewResource(spinner_dd_item);
    magnitudeSpinner.setAdapter(mAdapter);
  }
```

6. Добавьте публичные строковые значения, которые будут использоваться для определения ключей Общих настроек и где будут храниться значения каждой настройки. Обновите метод onCreate для получения именных настроек и вызовите метод updateUIFromPreferences. Он использует метод get<тип> из объекта Общих настроек, чтобы получать значения каждой настройки и применять их к текущему пользовательскому интерфейсу.

   Используйте объект Общих настроек по умолчанию, чтобы сохранить значения ваших настроек:

```
public static final String PREF_AUTO_UPDATE = "PREF_AUTO_UPDATE";
public static final String PREF_MIN_MAG = "PREF_MIN_MAG";
public static final String PREF_UPDATE_FREQ = "PREF_UPDATE_FREQ";

SharedPreferences prefs;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
```

```
  setContentView(R.layout.preferences);

  updateFreqSpinner = (Spinner)findViewById(R.id.spinner_update_freq);
  magnitudeSpinner = (Spinner)findViewById(R.id.spinner_quake_mag);
  autoUpdate = (CheckBox)findViewById(R.id.checkbox_auto_update);

  populateSpinners();

  Context context = getApplicationContext();
  prefs = PreferenceManager.getDefaultSharedPreferences(context);
  updateUIFromPreferences();
}

private void updateUIFromPreferences() {
  boolean autoUpChecked = prefs.getBoolean(PREF_AUTO_UPDATE, false);
  int updateFreqIndex = prefs.getInt(PREF_UPDATE_FREQ, 2);
  int minMagIndex = prefs.getInt(PREF_MIN_MAG, 0);

  updateFreqSpinner.setSelection(updateFreqIndex);
  magnitudeSpinner.setSelection(minMagIndex);
  autoUpdate.setChecked(autoUpChecked);
}
```

7. Все еще находясь в методе onCreate, добавьте обработчики событий для кнопок OK и Cancel. При нажатии кнопки Cancel Активность должна закрываться, в то время как кнопка OK сперва вызывает метод savePreferences:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.preferences);
  updateFreqSpinner = (Spinner)findViewById(R.id.spinner_update_freq);
  magnitudeSpinner = (Spinner)findViewById(R.id.spinner_quake_mag);
  autoUpdate = (CheckBox)findViewById(R.id.checkbox_auto_update);

  populateSpinners();

  Context context = getApplicationContext();
  prefs = PreferenceManager.getDefaultSharedPreferences(context);
  updateUIFromPreferences();

  Button okButton = (Button) findViewById(R.id.okButton);
  okButton.setOnClickListener(new View.OnClickListener() {

    public void onClick(View view) {
      savePreferences();
      Preferences.this.setResult(RESULT_OK);
      finish();
    }
  });

  Button cancelButton = (Button) findViewById(R.id.cancelButton);
  cancelButton.setOnClickListener(new View.OnClickListener() {

    public void onClick(View view) {
      Preferences.this.setResult(RESULT_CANCELED);
```

```
      finish();
    }
  });
}

private void savePreferences() {
}
```

8. Заполните кодом метод savePreferences, чтобы записывать текущие настройки, основанные на выборе установок в пользовательском интерфейсе, в объект Общих настроек:

```
private void savePreferences() {
  int updateIndex = updateFreqSpinner.getSelectedItemPosition();
  int minMagIndex = magnitudeSpinner.getSelectedItemPosition();
  boolean autoUpdateChecked = autoUpdate.isChecked();

  Editor editor = prefs.edit();
  editor.putBoolean(PREF_AUTO_UPDATE, autoUpdateChecked);
  editor.putInt(PREF_UPDATE_FREQ, updateIndex);
  editor.putInt(PREF_MIN_MAG, minMagIndex);
  editor.commit();
}
```

9. Этим заканчивается создание Активности Preferences. Сделайте ее доступной для приложения, добавив в манифест строки:

```
<activity android:name=".Preferences"
          android:label="Earthquake Preferences">
</activity>
```

10. Теперь вернитесь к Активности Earthquake и добавьте поддержку нового файла Общих настроек, а также пункт меню, при выборе которого отображается Активность Preferences. Начните с добавления нового пункта меню. Расширьте метод onCreateOptionsMenu, внеся новый пункт, открывающий Активность с настройками:

```
static final private int MENU_PREFERENCES = Menu.FIRST+1;

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  menu.add(0, MENU_UPDATE, Menu.NONE, R.string.menu_update);
  menu.add(0, MENU_PREFERENCES, Menu.NONE, R.string.menu_preferences);

  return true;
}
```

11. Измените метод onOptionsItemSelected, чтобы отобразить Активность Preferences при выборе нового пункта меню. Создайте явное Намерение и передайте его в виде параметра методу startActivityForResult. Это

запустит экран с настройками и оповестит класс Earthquake о сохранении настроек через обработчик onActivityResult:

```
private static final int SHOW_PREFERENCES = 1;

public boolean onOptionsItemSelected(MenuItem item) {

  super.onOptionsItemSelected(item);

  switch (item.getItemId()) {
    case (MENU_UPDATE): {
      refreshEarthquakes();
      return true;
    }
    case (MENU_PREFERENCES): {
      Intent i = new Intent(this, Preferences.class);
      startActivityForResult(i, SHOW_PREFERENCES);
      return true;
    }
  }
  return false;
}
```

12. Запустите ваше приложение и выберите пункт Preferences в меню Активности. Активность Preferences должна отобразиться, как показано на рис. 6.1.

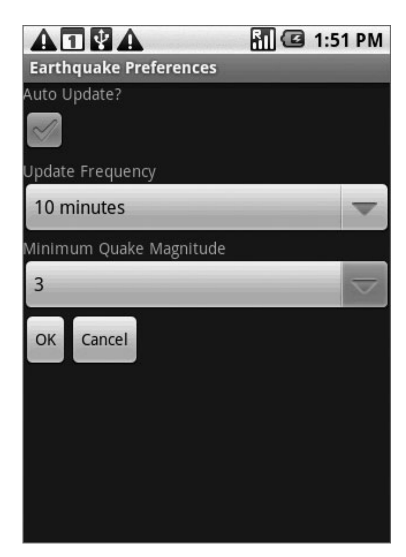


Рис. 6.1.

13. Осталось применить настройки к функциональности приложения. Реализацию автоматических обновлений оставим до главы 9, когда вы научитесь использовать сервисы и фоновые потоки. А пока можете применить фильтр с магнитудами с помощью фреймворка.

Начните с создания нового метода updateFromPreferences, который берет значения из Общих настроек и создает экземпляры переменных для каждого из них:

```
int minimumMagnitude = 0;
boolean autoUpdate = false;
int updateFreq = 0;

private void updateFromPreferences() {
  Context context = getApplicationContext();
  SharedPreferences prefs =
    PreferenceManager.getDefaultSharedPreferences(context);

  int minMagIndex = prefs.getInt(Preferences.PREF_MIN_MAG, 0);
  if (minMagIndex < 0)
    minMagIndex = 0;

  int freqIndex = prefs.getInt(Preferences.PREF_UPDATE_FREQ, 0);
  if (freqIndex < 0)
    freqIndex = 0;

  autoUpdate = prefs.getBoolean(Preferences.PREF_AUTO_UPDATE, false);

  Resources r = getResources();
  // Получите значения вариантов из массива.
  int[] minMagValues = r.getIntArray(R.array.magnitude);
  int[] freqValues = r.getIntArray(R.array.update_freq_values);

  // Преобразуйте значения в целочисленный тип.
  minimumMagnitude = minMagValues[minMagIndex];
  updateFreq = freqValues[freqIndex];
}
```

14. Примените фильтр магнитуд, обновив метод addNewQuake, чтобы проверить новую магнитуду землетрясения, прежде чем добавить ее в список:

```
private void addNewQuake(Quake _quake) {
  if (_quake.getMagnitude() > minimumMagnitude) {
    // Добавьте новое землетрясение в наш список.
    earthquakes.add(_quake);

    // Оповестите адаптер массива об изменении
    aa.notifyDataSetChanged();
  }
}
```

15. Переопределите обработчик onActivityResult, добавив вызов метода updateFromPreferences, чтобы обновлять список землетрясений при каждом сохранении изменений в Активности Preferences:

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

  if (requestCode == SHOW_PREFERENCES)
    if (resultCode == Activity.RESULT_OK) {
      updateFromPreferences();
```

```
    refreshEarthquakes();
  }
}
```

16. Наконец, вызовите updateFromPreferences в методе onCreate (перед вызовом refreshEarthquakes), чтобы настройки применялись во время запуска Активности:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  earthquakeListView = (ListView)this.findViewById(R.id.earthquakeListView);

  earthquakeListView.setOnItemClickListener(new OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView _av, View _v, int _index, long arg3) {
      selectedQuake = earthquakes.get(_index);
      showDialog(QUAKE_DIALOG);
    }
  });

  int layoutID = android.R.layout.simple_list_item_1;
  aa = new ArrayAdapter<Quake>(this, layoutID , earthquakes);
  earthquakeListView.setAdapter(aa);

  updateFromPreferences();
  refreshEarthquakes();
}
```

Знакомство с Активностью настроек и фреймворком для их создания

Для создания Экранов настроек в системном стиле Android предлагает фреймворк, основанный на XML. Используя его, вы можете быть уверены, что Активность с настройками станет выглядеть и вести себя так же, как аналогичная Активность в системе или сторонних приложениях. Из этого следуют два явных преимущества:

- пользователи знакомы с разметкой и принципом использования Экрана настроек в вашем приложении;
- вы можете интегрировать Экраны настроек из других приложений (включая системные настройки, такие как настройки местоположения) в Экраны настроек собственного приложения.

Фреймворк для создания Активности настроек (Preference Activity) состоит из трех частей.

- **Разметка экрана.** Файл XML, который определяет иерархию в вашей Активности настроек. В нем перечисляются элементы управления, которые нужно отобразить, допустимые значения и ключи Общих

настроек, используемые для каждого элемента управления в пользовательском интерфейсе.

- **Активность настроек.** Расширение класса PreferenceActivity, которое будет использоваться для размещения экранов с настройками вашего приложения.
- **Прослушивание изменений в Общих настройках.** Реализация класса onSharedPreferenceChangeListener, который предназначен для отслеживания изменений, вносимых в Общие настройки.

Фреймворк для создания Активности настроек — мощный инструмент, с помощью которого можно проектировать полностью настраиваемые динамические экраны с настройками. Весь спектр возможностей, которые предоставляет этот фреймворк, невозможно описать в рамках данной книги. Тем не менее в следующих разделах показывается, как создавать и использовать каждый из компонентов, описанных ранее.

Создание разметки Экрана настроек в формате XML

Одна из наиболее важных частей Активности настроек — XML-разметка. В отличии от обычной разметки пользовательского интерфейса определения настроек хранятся в каталоге ресурсов res/xml.

Принципиально не отличаясь от разметки пользовательского интерфейса, описанной в главе 4, разметка Экрана настроек использует особый набор элементов управления, разработанный специально для создания экранов с настройками (как в случае с системными настройками). Эти «родные» виджеты настроек описаны в следующем разделе.Каждая разметка настроек определяется в иерархическом виде, начиная с единственного элемента PreferenceScreen:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
</PreferenceScreen>
```

Вы можете добавить дополнительные элементы Экрана настроек, каждый из которых отобразится в виде элемента списка, и по щелчку на нем выводится новый экран.

Внутрь каждого Экрана настроек вы можете добавлять комбинацию элементов PreferenceCategory и Preference<элемент управления>. Элементы Категории настроек (Preference Category), показанные в следующем фрагменте кода, разбивают каждый Экран настроек на подкатегории с помощью заголовочных разделителей:

```
<PreferenceCategory
  android:title="My Preference Category"/>
</PreferenceCategory>
```

На рис. 6.2 показаны категории SIM card lock, Passwords и Credential storage, использованные в Экране настроек Location & security.

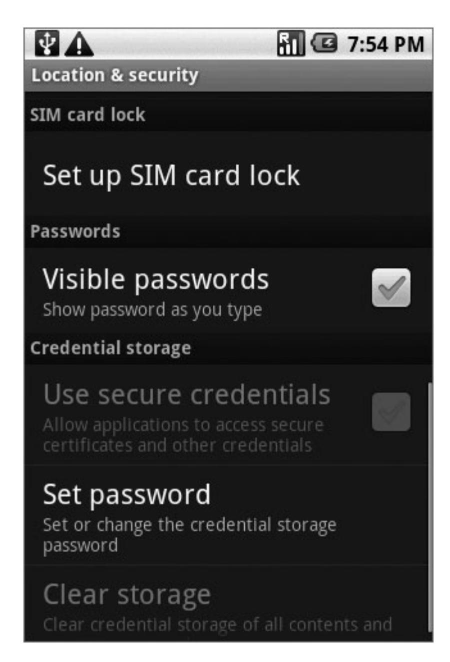


Рис. 6.2.

Все, что осталось, — добавить элементы управления настройками, необходимые для установки настроек приложения. Хотя набор атрибутов, доступных для каждого конкретного элемента управления, разный, все они включают:

- `android:key` — ключ Общих настроек, которому будет соответствовать записанное значение;
- `android:title` — текст, отображаемый для краткого описания настройки;
- `android:summary` — более длинное описание ниже заглавия, отображаемое шрифтом меньшего размера;
- `android:defaultValue` — значение по умолчанию, которое будет отображено (и выбрано), если для данного ключа не было назначено другого значения.

Листинг 6.3 показывает пример Экрана настроек, включающего Категорию настроек и элемент CheckBox.

Листинг 6.3. Простой экран с Общими настройками

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory
    android:title="My Preference Category"/>
```

Продолжение ⇗

Листинг 6.3 (продолжение)

```
      <CheckBoxPreference
        android:key="PREF_CHECK_BOX"
        android:title="Check Box Preference"
        android:summary="Check Box Preference Description"
        android:defaultValue="true"
      />
    </PreferenceCategory>
  </PreferenceScreen>
```

Этот Экран настроек выглядит, как показано на рис. 6.3.

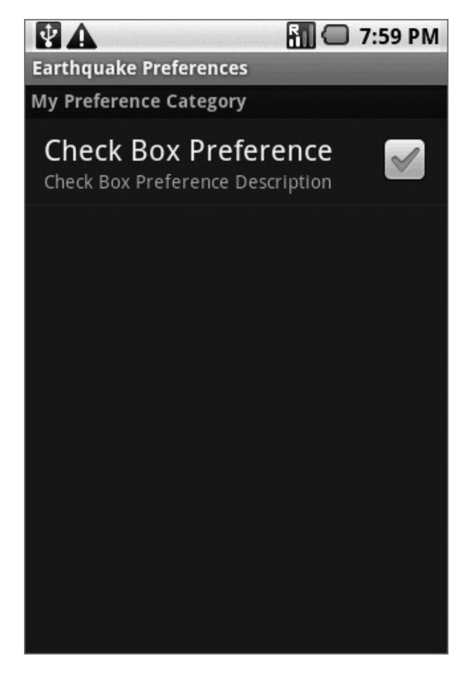


Рис. 6.3.

Стандартные элементы управления настройками

Android включает несколько элементов управления настройками, которые позволяют создать собственные Экраны настроек.

- CheckBoxPreference. Стандартный элемент управления типа CheckBox для настроек. Используется для установки значения настроек true или false.
- EditTextPreference. Позволяет пользователям вводить строковые значения в качестве настройки. При выборе этого элемента отображается окно ввода текста.
- ListPreference. Эквивалент Spinner для настроек. При выборе этого элемента отображается диалоговое окно, содержащее список значений,

одно из которых нужно выбрать. Вы можете указывать разные массивы, содержащие текст для отображения, а также значения для выбора.

- RingtonePreference. Специфический элемент, представляющий собой список доступных рингтонов, которые пользователь может выбрать. Это особенно полезно, когда создаете экран для изменения настроек уведомлений.

Каждый из этих элементов настроек может использоваться для построения иерархии вашего Экрана настроек. В качестве альтернативного варианта с помощью наследования класса Preference (или любого из его подклассов) вы можете создать собственный специальный элемент управления.

Более подробную информацию можно найти в документации по Android на странице http://developer.android.com/reference/android/preference/Preference.html.

Использование Намерений для импорта системных Экранов настроек

Как и в случае с собственными Экранами настроек, иерархии настроек могут включать Экраны настроек из других приложений (то же относится и к системным Экранам настроек).

Вы можете вызвать любую Активность в рамках вашего Экрана настроек с помощью Намерений. Если добавить узел Намерения в любой элемент Экрана настроек, система интерпретирует это как запрос на вызов метода startActivity с указанным действием.

Это может быть особенно полезно для добавления ссылок на соответствующие системные Экраны настроек в настройки вашего собственного приложения. В следующем фрагменте добавляется ссылка на системные настройки экрана:

```
<PreferenceScreen
  android:title="Intent preference"
  android:summary="System preference imported using an intent">
  <intent android:action="android.settings.DISPLAY_SETTINGS "/>
</PreferenceScreen>
```

Класс android.provider.Settings включает несколько констант android.settings.*, которые можно использовать для вызова экранов системных настроек.Для того чтобы ваши собственные Экраны настроек были доступны для вызова с помощью данного способа, нужно просто добавить Фильтр намерений в запись манифеста для Активности настроек приложения (подробно описывается в следующем разделе):

```
<activity android:name=".UserPreferences" android:label="Earthquake
Preferences">
  <intent-filter>
    <action android:name="com.paad.myapp.ACTION_USER_PREFERENCE" />
  </intent-filter>
</activity>
```

Знакомство с Активностью для управления настройками

Класс PreferenceActivity используется для хранения иерархии настроек, которые заданы с помощью файла XML. Чтобы создать новую Активность настроек, наследуйте класс PreferenceActivity, как показано ниже:

```
public class MyPreferenceActivity extends PreferenceActivity {
}
```

Чтобы заполнить настройки, переопределите обработчик onCreate и вызовите метод addPreferencesFromResource, как показано в следующем фрагменте:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  addPreferencesFromResource(R.xml.preferences);
}
```

Как и любая другая, Активность настроек должна быть включена в манифест приложения:

```
<activity android:name=".MyPreferenceActivity"
          android:label="My Preferences">
</activity>
```

Это все, что требуется для реализации простой Активности настроек. Чтобы отобразить настройки приложения, хранящиеся в этой Активности, вызовите метод startActivity или startActivityForResult:

```
Intent i = new Intent(this, MyPreferenceActivity.class);
startActivityForResult(i, SHOW_PREFERENCES);
```

Поиск и использование Экрана настроек с Общими настройками

Значения Общих настроек, записанные для пунктов, которые присутствуют в Активности настроек, хранятся в Контексте (Context) приложения. Это позволяет любому компоненту приложения, включая Активности, Сервисы и Приемники широковещательных намерений, получать доступ к значениям, как показано в следующем фрагменте:

```
Context context = getApplicationContext();
SharedPreferences prefs =
                 PreferenceManager.getDefaultSharedPreferences(context);
// TODO Получать значения с помощью методов get<тип>.
```

Введение в отслеживание изменений в Общих настройках

Класс onSharedPreferenceChangeListener может быть реализован для вызова callback-метода в момент добавления, удаления или изменения конкретной Общей настройки.

Это особенно удобно в случае с Активностями и Сервисами, которые используют фреймворк Общих настроек для установки настроек приложения. Используя этот обработчик, компоненты вашего приложения могут следить за изменениями в настройках, установленных пользователем, и обновлять пользовательский интерфейс или корректировать поведение программы, если потребуется.

Зарегистрируйте обработчик onSharedPreferenceChangeListener, применив Общие настройки, за которыми вы хотите следить. Реализация onSharedPreferenceChanged показана в листинге 6.4.

Листинг 6.4. Реализация каркаса onSharedPreferenceChanged

```
public class MyActivity extends Activity implements
  OnSharedPreferenceChangeListener {

  @Override
  public void onCreate(Bundle SavedInstanceState) {
    // Регистрируем этот OnSharedPreferenceChangeListener
    Context context = getApplicationContext();
    SharedPreferences prefs =
      PreferenceManager.getDefaultSharedPreferences(context);
    prefs.registerOnSharedPreferenceChangeListener(this);
  }

  public void onSharedPreferenceChanged(SharedPreferences prefs, String
key) {
    // TODO Проверять общие настройки, ключевые параметры и изменять UI
    // или поведение приложения, если потребуется.
  }
}
```

Создание стандартной Активности настроек для приложения Earthquake Viewer

Ранее в этой главе вы создали собственную Активность, чтобы пользователь мог вносить изменения в настройки приложения Earthquake Viewer. В этом примере[1] предстоит заменить эту Активность с помощью фреймворка для создания стандартных настроек приложения, описанного в предыдущем разделе.

1. Начните с создания нового каталога с ресурсами в формате XML — res/xml. В нем создайте новый файл `userpreferences.xml`. Он будет содержать описание пользовательского интерфейса для настроек вашего приложения. Используйте те же элементы управления и ресурсы данных, что и в предыдущей Активности, но на этот раз создайте их с помощью фреймворка.

[1] Все фрагменты кода в этом примере — часть проекта Earthquake из главы 6, их можно загрузить с сайта Wrox.com.

Убедитесь, что используете те ключи настроек, которые определили ранее.

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
  <CheckBoxPreference
    android:key="PREF_AUTO_UPDATE"
    android:title="Auto refresh"
    android:summary="Select to turn on automatic updating"
    android:defaultValue="true"
  />
  <ListPreference
    android:key="PREF_UPDATE_FREQ"
    android:title="Refresh frequency"
    android:summary="Frequency at which to refresh earthquake list"
    android:entries="@array/update_freq_options"
    android:entryValues="@array/update_freq_values"
    android:dialogTitle="Refresh frequency"
    android:defaultValue="60"
  />
  <ListPreference
    android:key="PREF_MIN_MAG"
    android:title="Minimum magnitude"
    android:summary="Select the minimum magnitude earthquake to report"
    android:entries="@array/magnitude_options"
    android:entryValues="@array/magnitude"
    android:dialogTitle="Magnitude"
    android:defaultValue="3"
  />
</PreferenceScreen>
```

2. Откройте Активность настроек и измените класс, от которого она наследуется, на PreferenceActivity:

```
public class UserPreferences extends PreferenceActivity
```

3. Активность настроек будет вмещать элементы управления для пользовательского интерфейса, так что вы можете удалить переменные, которые требовались для хранения объектов CheckBox и Spinner. Вы также можете убрать методы populateSpinners, updateUIFromPreferences и savePreferences.4. Отредактируйте метод onCreate. Удалите все ссылки на элементы управления, а также на кнопки OK и Cancel. Вместо этого загрузите файл настроек пользовательского интерфейса, который создали в пункте 1:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  addPreferencesFromResource(R.xml.preferences);
}
```

4. Теперь, когда вы запустите свое приложение и выберете пункт меню Preferences, появится новый стандартный экран с настройками, как показано на рис. 6.4.

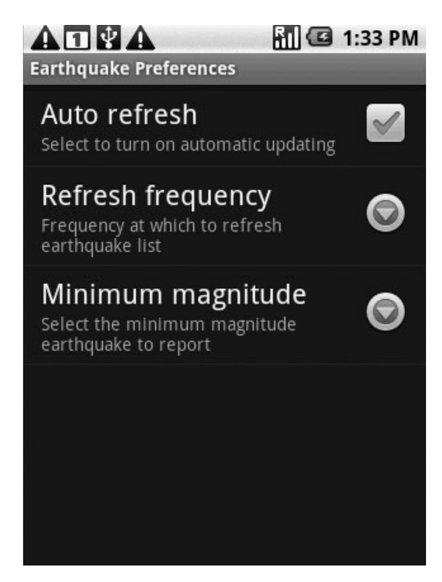


Рис. 6.4.

Сохранение состояния Активности

Если вы хотите сохранить информацию, которая принадлежит Активности и не должна быть доступна другим компонентам (например, переменным экземпляра класса), вы можете вызвать метод Activity.getPreferences() без указания названия Общих настроек. Доступ к возвращенному ассоциативному массиву Общих настроек ограничен Активностью, из которой он был вызван. Каждая Активность поддерживает только один безымянный объект Общих настроек.

В листинге 6.5 показано, как использовать приватные Общие настройки Активности.

Листинг 6.5. Сохранение состояния Активности

```
protected void saveActivityPreferences() {
  // Создайте или извлеките объект настроек Активности.
  SharedPreferences activityPreferences = getPreferences(Activity.MODE_
PRIVATE);

  // Извлеките редактор, чтобы изменить Общие настройки.
  SharedPreferences.Editor editor = activityPreferences.edit();

  // Извлеките Представление.
  TextView myTextView = (TextView)findViewById(R.id.myTextView);

  // Запишите новые значения примитивных типов в объект Общих настроек.
  editor.putString("currentTextValue", myTextView.getText().toString());

  // Сохраните изменения.
  editor.commit();
}
```

Сохранение и восстановление состояния экземпляра класса

Android предлагает специальный тип Общих настроек для сохранения переменных экземпляра Активности.

С помощью переопределения обработчика событий Активности onSaveInstanceState вы можете вызвать его параметр Bundle для сохранения переменных экземпляра пользовательского интерфейса. Прежде чем передавать измененный параметр Bundle в обработчик родительского класса, как показано в листинге 6.6, сохраните значения с помощью методов get и put, как и в случае с Общими настройками.

Листинг 6.6. Сохранение состояния экземпляра Активности

```
private static final String TEXTVIEW_STATE_KEY = "TEXTVIEW_STATE_KEY";

@Override
public void onSaveInstanceState(Bundle saveInstanceState) {
  // Извлеките Представление
  TextView myTextView = (TextView)findViewById(R.id.myTextView);

  // Сохраните его состояние
  saveInstanceState.putString(TEXTVIEW_STATE_KEY, myTextView.getText().
toString());
  super.onSaveInstanceState(saveInstanceState);
}
```

Этот обработчик будет срабатывать всякий раз, когда жизненный цикл Активности начнет подходить к концу, но только в том случае, если ее работа не будет завершена явно (при вызове метода finish). Вследствие этого обработчик используется для проверки целостности состояния Активности между активными жизненными циклами одиночной пользовательской сессии.

Сохраненный параметр Bundle передается методам onRestoreInstanceState и onCreate, если приложение принудительно перезапускается на протяжении сессии. В листинге 6.7 показано, как извлечь значения из этого параметра и использовать их для обновления состояния экземпляра Активности.

Листинг 6.7. Восстановление состояния экземпляра Активности

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  TextView myTextView = (TextView)findViewById(R.id.myTextView);

  String text = "";
```

```
  if (savedInstanceState != null && savedInstanceState.
containsKey(TEXTVIEW_STATE_KEY))

    text = savedInstanceState.getString(TEXTVIEW_STATE_KEY);

  myTextView.setText(text);
}
```

ПРИМЕЧАНИЕ

Надо помнить, что обработчик onSaveInstanceState вызывается только тогда, когда Активность переходит в пассивное состояние, а не когда она закрывается при вызове метода finish или пользователь нажимает кнопку Назад.

Сохранение состояния Активности из приложения To-Do List

На данном этапе каждый раз, когда приложение To-Do List перезапускается, все элементы списка теряются, а любой текст, введенный в текстовое поле, сбрасывается. В этом примере[1] вы начнете сохранять состояние приложения To-Do List между сессиями.

Состояние экземпляра в Активности To-Do List включает три переменные, которые помогают понять:

- был ли добавлен новый элемент;
- какой текст находится в новом текстовом поле;
- какой элемент списка сейчас выбран.

Используя Общие настройки Активности по умолчанию, можно хранить каждое из этих значений и обновлять пользовательский интерфейс, когда Активность перезапустят.

ПРИМЕЧАНИЕ

В следующей главе вы узнаете, как использовать базу данных SQLite для хранения элементов списка. Этот пример — первый шаг: показывается, как оставить у пользователя положительные впечатления от приложения, сохраняя данные экземпляра Активности.

1. Начните с добавления статических строковых переменных, которые будут использоваться в качестве ключей для настроек:

```
private static final String TEXT_ENTRY_KEY = "TEXT_ENTRY_KEY";
private static final String ADDING_ITEM_KEY = "ADDING_ITEM_KEY";
private static final String SELECTED_INDEX_KEY = "SELECTED_INDEX_KEY";
```

[1] Все фрагменты кода в этом примере — часть проекта Todo List из главы 6, их можно загрузить с сайта Wrox.com.

2. Переопределите метод onPause. Получите приватный объект Общих настроек Активности и объект Editor.Используя ключи, созданные в пункте 1, сохраните переменные экземпляра Активности, учитывая добавление нового элемента, а также сохраните текст из поля ввода new item:

```
@Override
protected void onPause(){
  super.onPause();

  // Получите объект настроек Активности.
  SharedPreferences uiState = getPreferences(0);
  // Получите доступ к редактору настроек.
  SharedPreferences.Editor editor = uiState.edit();

  // Добавьте в качестве настроек значения состояния пользовательского
интерфейса.
  editor.putString(TEXT_ENTRY_KEY, myEditText.getText().toString());
  editor.putBoolean(ADDING_ITEM_KEY, addingNew);
  // Сохраните настройки.
  editor.commit();
}
```

3. Создайте метод restoreUIState, применяющий значения экземпляра Активности, которые вы записали в предыдущем шаге, к приложению во время его перезапуска. Измените метод onCreate, добавив в самом конце вызов метода restoreUIState:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  [ ... Ранее написанный код ... ]
  restoreUIState();
}

private void restoreUIState() {
  // Получите объект настроек Активности.
  SharedPreferences settings = getPreferences(Activity.MODE_PRIVATE);

  // Извлеките значения состояния пользовательского интерфейса, укажите
значения по умолчанию.
  String text = settings.getString(TEXT_ENTRY_KEY, "");
  Boolean adding = settings.getBoolean(ADDING_ITEM_KEY, false);

  // Восстановите предыдущее состояние пользовательского интерфейса.
  if (adding) {
    addNewItem();
    myEditText.setText(text);
  }
}
```

4. Запишите индекс выбранного элемента списка с помощью механизма onSaveInstanceState/onRestoreInstanceState. Далее он будет сохранен и восстановлен только в том случае, если работа приложения завершится без явного участия пользователя:

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
  savedInstanceState.putInt(SELECTED_INDEX_KEY, myListView.
getSelectedItemPosition());
  super.onSaveInstanceState(saveInstanceState);
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
  int pos = -1;

  if (savedInstanceState != null)
    if (savedInstanceState.containsKey(SELECTED_INDEX_KEY))
      pos = savedInstanceState.getInt(SELECTED_INDEX_KEY, -1);

  myListView.setSelection(pos);
}
```

Теперь при запуске приложения To-Do List вы должны увидеть, что состояние пользовательского интерфейса сохраняется между сессиями. Тем не менее приложение все еще не сохраняет элементы списка — вы добавите эту важную часть функционала в следующей главе.

Сохранение и загрузка файлов

Использование Общих настроек или базы данных для хранения информации из вашего приложения — хорошее решение, но случается, что хочется работать с файлами напрямую, а не полагаться на механизмы управления, предоставляемые платформой Android.

Наряду со стандартными классами и методами Java, отвечающими за ввод/вывод, Android предоставляет методы openFileInput и openFileOutput для упрощения чтения и записи потоков, относящихся к локальным файлам, как показано в листинге 6.8.

Листинг 6.8. Сохранение и загрузка файлов

```
String FILE_NAME = "tempfile.tmp";

// Создайте новый исходящий файловый поток, который является приватным
// по отношению к данному приложению.
FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
// Создайте новый входящий файловый поток.
FileInputStream fis = openFileInput(FILE_NAME);
```

Эти методы поддерживают только те файлы, которые находятся в каталоге с текущим приложением; указание разделителей в пути приведет к выбросу исключения.Если при создании FileOutputStream файл, который вы указали, не существует, Android создаст его для вас. По умолчанию имеющиеся файлы перезаписываются. Чтобы добавить данные в конец существующего файла, установите режим Context.MODE_APPEND. По умолчанию файлы, созданные с помощью метода openFileOutput, принадлежат

только текущему приложению, другие приложения не будут иметь к нему доступ. Стандартный способ открыть доступ к файлу разным приложениям — использование класса ContentProvider. Еще один вариант — указать при создании файла режим Context.MODE_WORLD_READABLE или Context.MODE_WORLD_WRITEABLE, как показано в следующем фрагменте кода:

```
String OUTPUT_FILE = "publicCopy.txt";
FileOutputStream fos = openFileOutput(OUTPUT_FILE, Context.MODE_WORLD_
WRITEABLE);
```

Включение статических файлов в качестве ресурсов

Если для приложения требуются ресурсы в виде внешних файлов, можете включить их в дистрибутив, разместив в каталоге res/raw дерева вашего проекта. Чтобы получить доступ к ресурсам, предназначенным только для чтения, вызовите метод openRawResource, принадлежащий объекту Resource приложения. Таким образом, вы получите объект InputStream, основанный на указанном файле. В качестве имени переменной, принадлежащей R.raw, задайте имя файла (без расширения), как показано в следующем фрагменте:

```
Resources myResources = getResources();
InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```

Добавление файлов к вашему дереву ресурсов можно назвать хорошей альтернативой при использовании больших, заведомо существующих Источников данных (таких, как словари), которые нежелательно (или невозможно) конвертировать в формат базы данных Android. Механизм ресурсов в Android позволяет указывать альтернативные файлы ресурсов для различных языков, местностей и конфигураций аппаратного обеспечения. Например, вы можете создать приложение, которое загружает разные ресурсы со словарями, основываясь на языковых настройках, указанных пользователем.

Инструменты для управления файлами

Android содержит некоторые базовые инструменты для управления файлами, которые помогут вам при работе с файловой системой. Многие из них находятся в стандартном пакете java.io.File.

Полный обзор инструментов управления файлами в Java выходит за рамки данной книги, но Android предлагает некоторые специализированные утилиты для этих целей, доступные через объект Context приложения.

- deleteFile. Позволяет удалять файлы, созданные текущим приложением.

- fileList. Возвращает массив строк, в котором содержатся все файлы, созданные текущим приложением.

Эти методы используются при очистке временных файлов, которые были оставлены приложением в результате непредвиденного или принудительного завершения работы.

Резюме

В этой главе вы узнали, как хранить простые данные в рамках приложения и как управлять файлами и настройками. После изучения того, как сохранять данные экземпляра Активности между сессиями, используя обработчики сохранения и восстановления состояния экземпляра класса, вы познакомились с Общими настройками и фреймворком для управления системными Экранами настроек. Они применялись для сохранения значений экземпляра приложения и пользовательских настроек, которые могли бы пригодиться для различных компонентов приложения.Вы также узнали, как:

- напрямую сохранять и загружать файлы, обращаясь непосредственно к файловой системе;
- включать в проект статические файлы в качестве внешних ресурсов.

В следующей главе научитесь хранить более сложную и структурированную информацию в рамках приложения. Наряду с методами, описанными в этой главе, Android предоставляет доступ к полнофункциональным реляционным базам данных (используя библиотеку базы данных SQLite), который может осуществляться из разных приложений с помощью Источников данных. И SQLite, и Источники данных будут рассмотрены в следующей главе.

Глава 7
БАЗЫ ДАННЫХ И ИСТОЧНИКИ ДАННЫХ

Содержание главы

- Создание баз данных и работа с SQLite.
- Использование Источников данных для распределения доступа к данным приложения.
- Запросы к Источникам данных.
- Использование Курсоров и ContentValues для записи и чтения информации из Источников данных.
- Различные аспекты проектирования баз данных.
- Знакомство со стандартными Источниками данных.
- Использование Источника данных, предоставляющего информацию о контактах.

В этой главе вы познакомитесь с библиотекой SQLite и узнаете, как использовать Источники данных для распределения доступа к структурированным данным внутри одной программы и между разными приложениями.

SQLite предлагает мощную библиотеку для работы с SQL — надежную и целостную абстрактную прослойку, над которой вы имеете полный контроль.

Источники данных предлагают общий интерфейс для доступа к любой информации путем отделения логики приложения от слоя, отвечающего за хранение данных.

По умолчанию доступ к базе данных ограничен приложением, которое ее создало. Источники данных предлагают стандартный интерфейс, через который вы можете обмениваться данными с другими приложениями (включая множество стандартных хранилищ информации).

Введение в базы данных на платформе Android

Структурированные данные в Android хранятся с использованием двух механизмов.

- **Базы данных SQLite.** Если нужно хранить управляемую, структурированную информацию, можете использовать библиотеку SQLite для работы с реляционными базами данных. Любое приложение может создавать свои собственные базы данных, над которыми оно будет иметь полный контроль.
- **Источники данных.** Источники данных предоставляют общий, строго определенный интерфейс для обмена данными.

Введение в базы данных SQLite

С помощью SQLite вы можете создавать для своего приложения независимые реляционные базы данных и использовать их для хранения и управления сложными, структурированными данными приложения.

Android хранит базы данных в каталоге /data/data/<имя_пакета>/databases на вашем устройстве (или эмуляторе). По умолчанию все базы данных приватные, доступ к ним могут получить только те приложения, которые их создали.

Проектирование базы данных — большая тема, которая заслуживает более детального описания, нежели это возможно в рамках данной книги. Стоит отметить, что наиболее удачные стандартные подходы по работе с базами данных применимы и к Android. В частности, когда вы создаете базы данных для устройств с ограниченными ресурсами (например, мобильных телефонов), важно нормализовать данные, чтобы уменьшить их избыточность.

Введение в Источники данных

Источники данных предоставляют интерфейс для публикации и потребления данных, основанный на простой адресной модели URI, используя схему `content://`. Этот механизм позволяет отделить логику приложения от данных, делая ваши программы нечувствительными к источникам, из которых поступает информация, путем скрытия базового источника с данными.

Общие Источники данных могут использоваться для получения результатов запросов, обновления или удаления существующих записей, а также для добавления новых. Любое приложение с соответствующими полномочиями может добавлять, удалять или изменять данные, принадлежащие другому приложению, включая стандартные базы данных Android.

Многие стандартные базы данных доступны в качестве Источников данных и могут использоваться сторонними приложениями. Сюда входят телефонные контакты, хранилище информации и другие стандартные базы данных, речь о которых пойдет далее в этой главе.

Публикуя собственные данные в виде Источников данных, вы получаете шанс (и даете его другим разработчикам) объединять и расширять их с помощью новых приложений.

Введение в SQLite

SQLite — это система управления реляционными базами данных, которая хорошо себя зарекомендовала. Ее основные характеристики:

- свободная;
- соответствует стандартам;
- легковесная;
- одноуровневая.

Данная система — это компактная библиотека, написанная на языке C, она составляет часть стека программного обеспечения, который поставляется вместе с Android.

Поскольку SQLite реализована в виде библиотеки, а не как отдельный исполняемый процесс, каждая база данных считается частью приложения, которое ее создало. Благодаря этому минимизируется число внешних зависимостей, уменьшаются задержки, упрощаются синхронизация и блокирование при выполнении транзакций.

SQLite зарекомендовала себя в качестве чрезвычайно надежной системы баз данных, которая используется во многих бытовых электронных устройствах, включая некоторые MP3-проигрыватели, iPhone и iPod Touch.

SQLite — мощная и легковесная, отличается от многих обычных движков баз данных отсутствием типизации каждого столбца. Это значит, что значения в столбце не обязаны иметь один и тот же тип. Вместо этого каждое значение типизируется отдельно для каждой строки. В связи с этим нет необходимости проверять типы при занесении или извлечении данных в контексте каждого столбца в строке.

Для более полного ознакомления с SQLite, включая ее сильные и слабые стороны, посетите официальный сайт http://www.sqlite.org/.

Курсоры и класс ContentValues

Класс ContentValues используется для добавления новых строк в таблицу. Каждый объект этого класса представляет собой одну строку таблицы и выглядит как ассоциативный массив с именами столбцов и значениями, которые им соответствуют.

В Android запросы к базе данных возвращают объекты класса Cursor. Вместо того чтобы извлекать данные и возвращать копию значений, Курсоры

ссылаются на результирующий набор исходных данных. Курсоры позволяют управлять текущей позицией (строкой) в результирующем наборе данных, возвращаемом при запросе.

Класс Cursor содержит немало возможностей для навигации (но не ограничивается только ими):

- moveToFirst — перемещает курсор на первую строку в результате запроса;
- moveToNext — перемещает курсор на следующую строку;
- moveToPrevious — перемещает курсор на предыдущую строку;
- getCount — возвращает количество строк в результирующем наборе данных;
- getColumnIndexOrThrow — возвращает индекс для столбца с указанным именем (выбрасывает исключение, если столбец с таким именем не существует);
- getColumnName — возвращает имя столбца с указанным индексом;
- getColumnNames — возвращает массив строк, содержащий имена всех столбцов в объекте Cursor;
- moveToPosition — перемещает курсор на указанную строку;
- getPosition — возвращает текущую позицию курсора.

Android предоставляет удобный механизм, который упрощает управление Курсорами в ваших Активностях. Метод startManagingCursor объединяет жизненный цикл объекта Cursor с Активностью, из которой он был вызван. Когда вы закончили работу с Курсором, вызовите метод stopManagingCursor, принцип работы которого понятен из его названия.

Далее в этой главе вы научитесь делать запросы к базе данных и извлекать из результирующих курсоров конкретные значения строк/столбцов.

Работа с базами данных SQLite

Создание вспомогательного класса для упрощения взаимодействий с базой данных — хорошее решение.

В этом разделе вы узнаете, как создавать класс Адаптера для базы данных. Эта абстрактная прослойка инкапсулирует взаимодействия с базой данных. Она предоставляет интуитивный, строго типизированный способ добавления, удаления и изменения элементов. Адаптер базы данных также должен обрабатывать запросы и переопределять методы для создания, открытия и закрытия базы данных. Он также может использоваться как удобное место для размещения статических констант, относящихся к базе данных (например, имен таблиц и столбцов).

В листинге 7.1 показан каркас для класса, реализующего стандартный Адаптер базы данных. В нем происходит наследование класса SQLiteOpenHelper (рассмотрен в следующем разделе), который нужен для упрощения открытия, создания и обновления базы данных.

Листинг 7.1. Каркас для реализации стандартного Адаптера базы данных

```
import android.content.Context;
import android.database.*;
import android.database.sqlite.*;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.util.Log;

public class MyDBAdapter {
  private static final String DATABASE_NAME = "myDatabase.db";
  private static final String DATABASE_TABLE = "mainTable";
  private static final int DATABASE_VERSION = 1;

  // Индекс (ключ) столбца, с которым мы будем работать.
  public static final String KEY_ID="_id";

  // Имя и индекс каждого столбца в вашей базе данных.
  public static final String KEY_NAME="name";
  public static final int NAME_COLUMN = 1;
  // TODO: Создать публичное поле для каждого столбца в вашей базе
данных.

  // Инструкция на языке SQL для создания новой базы данных.
  private static final String DATABASE_CREATE = "create table " +
    DATABASE_TABLE + " (" + KEY_ID +
    " integer primary key autoincrement, " +
    KEY_NAME + " text not null);";

  // Поле, хранящее экземпляр базы данных.
  private SQLiteDatabase db;
  // Объект Context приложения, которое использует базу данных.
  private final Context context;
  // Вспомогательный класс для открытия/обновления базы данных.
  private myDbHelper dbHelper;

  public MyDBAdapter(Context _context) {
    context = _context;
    dbHelper = new myDbHelper(context, DATABASE_NAME, null, DATABASE_
VERSION);
  }

  public MyDBAdapter open() throws SQLException {
    db = dbHelper.getWritableDatabase();
    return this;
  }

  public void close() {
    db.close();
  }

  public int insertEntry(MyObject _myObject) {
```

```
    // TODO: Создать новый объект ContentValues для представления нашей
    // строки и вставить его в базу данных.
    return index;
  }

  public boolean removeEntry(long _rowIndex) {
    return db.delete(DATABASE_TABLE, KEY_ID + "=" + _rowIndex, null) > 0;
  }

  public Cursor getAllEntries () {
    return db.query(DATABASE_TABLE, new String[] {KEY_ID, KEY_NAME},
                    null, null, null, null, null);
  }

  public MyObject getEntry(long _rowIndex) {
    // TODO: Возвратить курсор, ссылающийся на строку из базы данных,
    // и использовать значения для заполнения данными экземпляра
MyObject.
    return objectInstance;
  }

  public boolean updateEntry(long _rowIndex, MyObject _myObject) {
    // TODO: Создать экземпляр класса ContentValues, основанный на новом
    // объекте, и использовать его для обновления строки в базе данных.
    return true;
  }

  private static class myDbHelper extends SQLiteOpenHelper {

    public myDbHelper(Context context, String name,
                      CursorFactory factory, int version) {
      super(context, name, factory, version);
    }

    // Вызывается, когда база данных не найдена на носителе,
    // и вспомогательный класс должен создать новую.
    @Override
    public void onCreate(SQLiteDatabase _db) {
      _db.execSQL(DATABASE_CREATE);
    }

    // Вызывается при несоответствии версии базы данных, в следствие чего
    // версия базы данных на носителе нуждается в обновлении до текущей.
    @Override
    public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _
newVersion) {
      // Логируйте обновление версии.
      Log.w("TaskDBAdapter", "Upgrading from version " +
                     _oldVersion + " to " +
                     _newVersion + ", which will destroy all old data");

      // Приведите существующую базу данных в соответствие с новой
      // версией. Все предыдущие версии могут быть обработаны путем
      // сравнения значений _oldVersion и _newVersion.

      // Самый простой способ - удалить старую таблицу и создать новую.
      _db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
```

Продолжение ⇗

Листинг 7.1 (продолжение)

```
        // Create a new one.
        onCreate(_db);
      }
    }
  }
```

Знакомство с классом SQLiteOpenHelper

SQLiteOpenHelper — абстрактный класс, предназначенный для реализации наиболее эффективной модели, с помощью которой можно создавать, открывать и обновлять базы данных. При реализации этого вспомогательного класса от вас скрывается логика, на основе которой принимается решение о создании или обновлении базы данных перед ее открытием.

В листинге 7.1 показано, как наследовать класс SQLiteOpenHelper, переопределив конструктор, а также методы onCreate и onUpgrade, чтобы контролировать создание новой базы данных и обновление ее до последней версии соответственно.

ПРИМЕЧАНИЕ

В предыдущем примере обработчик onUpgrade просто удалял существующую таблицу и создавал новую. На практике лучшее решение — перенос существующих данных в новую таблицу.

Чтобы использовать реализацию вспомогательного класса, создайте новый экземпляр, передайте его конструктору контекст, имя базы данных, текущую версию и объект класса CursorFactory (если вы его используете).

Вызовите метод getReadableDatabase или getWritableDatabase, чтобы открыть и вернуть экземпляр базы данных, с которой мы имеем дело (он будет доступен как для чтения, так и для записи).

Вызов метода getWritableDatabase может завершиться неудачно из-за проблем с полномочиями или нехваткой места на диске, поэтому лучше предусмотреть откат к методу getReadableDatabase, как показано в листинге 7.2.

Листинг 7.2. Использование SQLiteOpenHelper для доступа к базе данных

```
dbHelper = new myDbHelper(context, DATABASE_NAME, null, DATABASE_
VERSION);

SQLiteDatabase db;
try {
  db = dbHelper.getWritableDatabase();
}
catch (SQLiteException ex){
  db = dbHelper.getReadableDatabase();
}
```

Внутри скрыта следующая логика: если база данных не существует, вспомогательный класс вызывает свой обработчик onCreate; если версия базы данных изменилась, вызовется обработчик onUpgrade. В любом случае вызов методов getWritableDatabase/getReadableDatabase, в зависимости от ситуации, вернет существующую, только что созданную или обновленную базу данных.

Открытие и создание баз данных без использования SQLiteHelper

Вы можете открывать и создавать базы данных без помощи класса SQLiteHelper, используя метод openOrCreateDatabase, принадлежащий объекту Context вашего приложения.

Получите доступ к базе данных в два шага. Сначала вызовите метод openOrCreateDatabase, чтобы создать новую базу данных. Затем из полученного экземпляра базы данных вызовите execSQL, чтобы выполнять команды на языке SQL, с помощью которых будут созданы таблицы и установлены отношения между ними. Весь этот процесс показан в листинге 7.3.

Листинг 7.3. Создание новой базы данных

```
private static final String DATABASE_NAME = "myDatabase.db";
private static final String DATABASE_TABLE = "mainTable";

private static final String DATABASE_CREATE =
  "create table " + DATABASE_TABLE + " ( _id integer primary key
autoincrement," +
  "column_one text not null);";

SQLiteDatabase myDatabase;

private void createDatabase() {
  myDatabase = openOrCreateDatabase(DATABASE_NAME, Context.MODE_PRIVATE,
null);
  myDatabase.execSQL(DATABASE_CREATE);
}
```

Особенности организации баз данных в Android

Существует несколько специфичных для Android особенностей, которые нужно учитывать при проектировании базы данных:

- файлы (например, графические или звуковые файлы), как правило, не хранятся в таблицах базы данных. Используйте строку для хранения пути к файлу, а еще лучше полный URI;
- хоть это и необязательно, но настоятельно рекомендуется, чтобы все таблицы имели поле с автоинкрементным ключом, который станет уникальным индексом для каждой строки. Если планируете делегировать

доступ к вашей таблице с помощью Источников данных, поле с уникальным ID обязательно.

Выполнение запросов к базе данных

Каждый запрос к базе данных возвращает объект Cursor. Это позволяет Android управлять ресурсами более эффективно, получая и освобождая значения строк и столбцов по запросу. Чтобы выполнить запрос к базе данных, используйте метод query, передавая ему:

- необязательное значение типа Boolean, которое определяет, должен ли результирующий набор данных содержать исключительно уникальные значения;
- имя базы данных, к которой нужно получить доступ;
- проекцию в виде массива строк, содержащего список столбцов, которые нужно включить в результирующий набор;
- оператор WHERE, определяющий те строки, что должны быть возвращены (можно использовать маски «?», которые заменятся переданными значениями для оператора SELECT);
- массив строк, выступающих в роли аргументов для оператора SELECT, которые заменят маски «?» в операторе WHERE;
- оператор GROUP BY, который определяет, каким образом будут сгруппированы результирующие строки;
- фильтр HAVING, который определяет, какие группы строк включать в результат, если использовался оператор GROUP BY;
- строку, описывающую порядок возвращения результирующих строк;
- необязательный строковой параметр, определяющий максимальное количество возвращаемых строк.

В листинге 7.4 показан фрагмент кода, который возвращает все или некоторые строки в заданной таблице.

Листинг 7.4. Выполнение запросов к базе данных

```
// Возвращает все строки для первого и третьего столбца, без повторений
String[] result_columns = new String[] {KEY_ID, KEY_COL1, KEY_COL3};

Cursor allRows = myDatabase.query(true, DATABASE_TABLE, result_columns,
                                  null, null, null, null, null, null);

// Возвращает все столбцы для строк, в которых третий столбец совпадает
// с заданным значением, а сами строки отсортированы по пятому столбцу.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor myResult = myDatabase.query(DATABASE_TABLE, null, where,
                                      null, null, null, order);
```

Извлечение результатов из объекта Cursor

Чтобы извлечь значения из результирующего Курсора, сперва необходимо воспользоваться методами moveTo<местоположение>, описанными ранее, чтобы поставить Курсор в правильную строку.

Далее используйте типизированные методы get<тип> (передавая им индексы столбцов), чтобы вернуть значение, хранимое в текущей строке для указанного столбца, как показано в следующем фрагменте:

```
String columnValue = myResult.getString(columnIndex);
```

ПРИМЕЧАНИЕ

Реализации баз данных должны предусматривать константы, которые предоставляют имена и/или индексы столбцов, имеющие легко узнаваемые названия, основанные на именах столбцов. Эти статические константы, как правило, используются внутри Адаптера базы данных.

В листинге 7.5 показано, как можно пройтись по результирующему объекту Cursor, извлекая и суммируя столбцы со значениями типа float.

Листинг 7.5. Извлечение значений из объекта Cursor

```
int GOLD_HOARDED_COLUMN = 2;
Cursor myGold = myDatabase.query("GoldHoards", null, null, null, null,
null, null);
float totalHoard = 0f;

// Убедитесь, что у нас есть как минимум одна строка.
if (myGold.moveToFirst()) {
  // Пройдитесь по каждой строке.
  do {
    float hoard = myGold.getFloat(GOLD_HOARDED_COLUMN);
    totalHoard += hoard;
  } while(myGold.moveToNext());
}

float averageHoard = totalHoard / myGold.getCount();
```

Поскольку столбцы в базах данных SQLite слабо типизированы, вы можете приводить отдельные значения к допустимым типам при необходимости. Например, значения, хранящиеся в виде натуральных чисел, можно прочитать как строки.

Добавление, обновление и удаление строк в таблице

Класс SQLiteDatabase предоставляет методы insert, delete и update, которые инкапсулируют операторы языка SQL, необходимые для данных действий. Кроме того, метод execSQL позволяет выполнять любой допустимый

код на языке SQL применимо к таблицам базы данных, если вы хотите провести эти (или любые другие) операции вручную.

Каждый раз, когда вы редактируете очередное значение из базы данных, нужно вызывать метод refreshQuery для всех Курсоров, которые имеют отношение к редактируемой таблице.

Вставка новых строк

Для создания новой строки понадобится объект ContentValues, точнее, его метод put, чтобы обеспечить данными каждый столбец. Вставьте новую строку, передавая в метод insert, вызванный в контексте нужной нам базы данных, имя таблицы и объект ContentValues, как показано в листинге 7.6.

Листинг 7.6. Вставка новых строк в базу данных

```
// Создайте новую строку со значениями для вставки.
ContentValues newValues = new ContentValues();

// Задайте значения для каждой строки.
newValues.put(COLUMN_NAME, newValue);
[ ... Повторите для каждого столбца ... ]

// Вставьте строку в вашу базу данных.
myDatabase.insert(DATABASE_TABLE, null, newValues);
```

Обновление строк в таблице

Обновление строк также происходит с помощью класса ContentValues.

Создайте новый объект ContentValues, используя метод put для вставки значений в каждый столбец, который вы хотите обновить. Вызовите метод update в контексте базы данных, передайте ему имя таблицы, обновленный объект ContentValues и оператор WHERE, указывающий на строку (строки), которую нужно обновить. Все это показано в листинге 7.7.

Листинг 7.7. Обновление строки в базе данных

```
// Определите содержимое обновленной строки.
ContentValues updatedValues = new ContentValues();

// Назначьте значения для каждой строки.
newValues.put(COLUMN_NAME, newValue);
[ ... Повторите для каждого столбца ... ]

String where = KEY_ID + "=" + rowId;

// Обновите строку с указанным индексом, используя новые значения.
myDatabase.update(DATABASE_TABLE, newValues, where, null);
```

Удаление строк из таблицы

Чтобы удалить строку, просто вызовите метод delete в контексте базы данных, указав имя таблицы и оператор WHERE. В результате вы получите строки, которые хотите удалить, как показано в листинге 7.8.

Листинг 7.8. Удаление строки из базы данных

```
myDatabase.delete(DATABASE_TABLE, KEY_ID + "=" + rowId, null);
```

Сохранение пунктов списка в приложении To-Do List

В главе 6 вы улучшили пример To-Do List, добавив сохранение состояния пользовательского интерфейса Активностей между сессиями. Но это была только половина работы. В следующем примере[1] вы создадите базу данных для сохранения пунктов списка.

1. Начните с создания нового класса ToDoDBAdapter. Он нужен для взаимодействия с вашей базой данных. Создайте приватные поля для хранения объекта SQLiteDatabase, а также Context, принадлежащего приложению. Добавьте конструктор, который принимает Context в качестве параметра и оформляет статические поля для имени и версии базы данных, а также для имени таблицы, в которой будут храниться пункты из списка.

```
package com.paad.todolist;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class ToDoDBAdapter {
  private static final String DATABASE_NAME = "todoList.db";
  private static final String DATABASE_TABLE = "todoItems";
  private static final int DATABASE_VERSION = 1;

  private SQLiteDatabase db;
  private final Context context;

  public ToDoDBAdapter(Context _context) {
    this.context = _context;
  }
}
```

[1] Все фрагменты кода в этом примере — часть проекта Todo List из главы 7, их можно загрузить с сайта Wrox.com.

2. Создайте публичные вспомогательные поля, которые определяют имена столбцов. Это упростит нахождение нужных столбцов при извлечении данных из результирующих Курсоров:

```
public static final String KEY_ID = "_id";
public static final String KEY_TASK = "task";
public static final String KEY_CREATION_DATE = "creation_date";
```

3. Создайте новый класс toDoDBOpenHelper внутри ToDoDBAdapter, наследующий SQLiteOpenHelper. Он нужен для упрощения работы с версиями вашей базы данных. Внутри него переопределите методы onCreate и onUpgrade для управления логикой создания и обновления базы данных.

```
private static class toDoDBOpenHelper extends SQLiteOpenHelper {

  public toDoDBOpenHelper(Context context, String name,
                        CursorFactory factory, int version) {
    super(context, name, factory, version);
  }

  // Конструкция на языке SQL для создания новой базы данных.
  private static final String DATABASE_CREATE = "create table " +
    DATABASE_TABLE + " (" + KEY_ID + " integer primary key
autoincrement, " +
    KEY_TASK + " text not null, " + KEY_CREATION_DATE + " long);";

  @Override
   public void onCreate(SQLiteDatabase _db) {
    _db.execSQL(DATABASE_CREATE);
  }

  @Override
  public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _
newVersion) {
    Log.w("TaskDBAdapter", "Upgrading from version " +
                      _oldVersion + " to " +
                      _newVersion + ", which will destroy all old data");

    // Удалите старую таблицу.
    _db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
    // Создайте новую.
    onCreate(_db);
  }
}
```

4. Добавьте приватное поле для класса ToDoDBAdapter, которое будет хранить экземпляр класса toDoDBOpenHelper, созданного в предыдущем пункте. Инициализируйте его внутри конструктора.

```
private toDoDBOpenHelper dbHelper;

public ToDoDBAdapter(Context _context) {
  this.context = _context;
```

```
  dbHelper = new toDoDBOpenHelper(context, DATABASE_NAME,
                                null, DATABASE_VERSION);
}
```

5. В этом же классе создайте методы open и close, которые икнапсулируют логику, отвечающую за открытие и закрытие базы данных. Начните с метода close: внутри него просто вызывается одноименный метод из объекта базы данных.

```
public void close() {
  db.close();
}
```

6. В методе open должен использоваться класс toDoDBOpenHelper. Вызовите метод getWritableDatabase, чтобы позволить этому вспомогательному классу обрабатывать создание базы данных и проверку ее версий. Предусмотрите выброс исключения при данном вызове, чтобы открыть базу данных в режиме для чтения, если режим для записи недоступен.

```
public void open() throws SQLiteException {
  try {
    db = dbHelper.getWritableDatabase();
  } catch (SQLiteException ex) {
    db = dbHelper.getReadableDatabase();
  }
}
```

7. Внесите строго типизированные методы для добавления, удаления и обновления элементов списка.

```
// Вставка нового элемента
public long insertTask(ToDoItem _task) {
  // Создайте новую строку со значениями, которые нужно вставить.
  ContentValues newTaskValues = new ContentValues();
  // Присвойте значения для каждой строки.
  newTaskValues.put(KEY_TASK, _task.getTask());
  newTaskValues.put(KEY_CREATION_DATE, _task.getCreated().getTime());
  // Вставьте строку.
  return db.insert(DATABASE_TABLE, null, newTaskValues);
}

// Удаление элемента с конкретным индексом
public boolean removeTask(long _rowIndex) {
  return db.delete(DATABASE_TABLE, KEY_ID + "=" + _rowIndex, null) > 0;
}

// Обновление элемента
public boolean updateTask(long _rowIndex, String _task) {
  ContentValues newValue = new ContentValues();
  newValue.put(KEY_TASK, _task);
  return db.update(DATABASE_TABLE, newValue, KEY_ID + "=" + _rowIndex,
null) > 0;
}
```

8. Добавьте вспомогательные методы для обработки запросов. Потребуются три метода: один возвращает все элементы, другой — конкретную строку в виде курсора, а третий — объект, приведенный к типу ToDoItem.

```
public Cursor getAllToDoItemsCursor() {
  return db.query(DATABASE_TABLE,
                  new String[] { KEY_ID, KEY_TASK, KEY_CREATION_DATE},
                  null, null, null, null, null);
}

public Cursor setCursorToToDoItem(long _rowIndex) throws SQLException {
  Cursor result = db.query(true, DATABASE_TABLE,
                           new String[] {KEY_ID, KEY_TASK},
                           KEY_ID + "=" + _rowIndex, null, null, null,
                           null, null);
  if ((result.getCount() == 0) || !result.moveToFirst()) {
    throw new SQLException("No to do items found for row: " + _rowIndex);
  }
  return result;
}

public ToDoItem getToDoItem(long _rowIndex) throws SQLException {
  Cursor cursor = db.query(true, DATABASE_TABLE,
                           new String[] {KEY_ID, KEY_TASK},
                           KEY_ID + "=" + _rowIndex, null, null, null,
                           null, null);
  if ((cursor.getCount() == 0) || !cursor.moveToFirst()) {
    throw new SQLException("No to do item found for row: " + _rowIndex);
  }

  String task = cursor.getString(TASK_COLUMN);
  long created = cursor.getLong(CREATION_DATE_COLUMN);

  ToDoItem result = new ToDoItem(task, new Date(created));
  return result;
}
```

9. На этом вспомогательный класс для работы с базой данных закончен. Вернитесь к Активности ToDoList и отредактируйте ее с учетом поддержки сохранения массива с заданиями. Начните с изменения метода onCreate, создав экземпляр класса toDoDBAdapter и открыв соединение с базой данных. Разместите также вызов метода-заглушки populateTodoList.

```
ToDoDBAdapter toDoDBAdapter;

public void onCreate(Bundle icicle) {
  [ ... ранее написанный код в методе onCreate ... ]

  toDoDBAdapter = new ToDoDBAdapter(this);

  // Откройте или создайте базу данных
```

```
    toDoDBAdapter.open();

    populateTodoList();
  }

  private void populateTodoList() { }
```

10. Создайте новое поле для хранения экземпляра Курсора, указывающего на все элементы в базе данных. Отредактируйте метод populateTodoList, используя экземпляр класса toDoDBAdapter, чтобы выполнять запросы к базе данных. Вызовите метод startManagingCursor, чтобы позволить Активности управлять объектом Cursor. Следует также вызвать метод updateArray — он понадобится для повторного заполнения массива со списком заданий с помощью Курсора.

```
  Cursor toDoListCursor;

  private void populateTodoList() {
    // Получите все элементы из базы данных.
    toDoListCursor = toDoDBAdapter. getAllToDoItemsCursor();
    startManagingCursor(toDoListCursor);

    // Обновите массив.
    updateArray();
  }

  private void updateArray() { }
```

11. Реализуйте метод updateArray, который будет обновлять массив со списком текущих заданий. Вызовите requery в контексте объекта Cursor, чтобы убедиться в его полной актуальности, затем очистите массив и пройдитесь по результирующему набору данных. Когда обновление закончится, вызовите метод notifyDataSetChanged из объекта ArrayAdapter.

```
  private void updateArray() {
    toDoListCursor.requery();

    todoItems.clear();

    if (toDoListCursor.moveToFirst())
      do {
        String task = toDoListCursor.getString(ToDoDBAdapter.TASK_COLUMN);
        long created = toDoListCursor.getLong(ToDoDBAdapter.CREATION_DATE_
  COLUMN);

        ToDoItem newItem = new ToDoItem(task, new Date(created));
        todoItems.add(0, newItem);
      } while(toDoListCursor.moveToNext());

    aa.notifyDataSetChanged();
  }
```

12. Чтобы свести все воедино, отредактируйте класс OnKeyListener, связанный с полем ввода в методе onCreate, и поправьте метод removeItem. Вместо изменения массива со списком заданий напрямую эти методы должны использовать toDoDBAdapter, чтобы добавлять и удалять элементы из базы данных.

    12.1. Начните с OnKeyListener. Вставьте новый элемент в базу данных и обновите массив.

```
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  setContentView(R.layout.main);

  myListView = (ListView)findViewById(R.id.myListView);
  myEditText = (EditText)findViewById(R.id.myEditText);

  todoItems = new ArrayList<ToDoItem>();
  int resID = R.layout.todolist_item;
  aa = new ToDoItemAdapter(this, resID, todoItems);
  myListView.setAdapter(aa);

  myEditText.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
      if (event.getAction() == KeyEvent.ACTION_DOWN)
        if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
          ToDoItem newItem = new ToDoItem(myEditText.getText().
toString());
          toDoDBAdapter.insertTask(newItem);
          updateArray();
          myEditText.setText("");
          aa.notifyDataSetChanged();
          cancelAdd();
          return true;
        }
      return false;
    }
  });

  registerForContextMenu(myListView);
  restoreUIState();

  toDoDBAdapter = new ToDoDBAdapter(this);

  // Откройте или создайте базу данных
  toDoDBAdapter.open();

  populateTodoList();
}
```

    12.2. Отредактируйте метод removeItem, чтобы удалять элементы из базы данных и обновлять массив.

```
private void removeItem(int _index) {
  // Элементы добавляются в объект ListView в обратном порядке, поэтому
инвертируйте индекс.
  toDoDBAdapter.removeTask(todoItems.size()-_index);
  updateArray();
}
```

13. Финальный шаг: переопределите метод onDestroy, принадлежащий вашей Активности, чтобы закрывать соединение с базой данных при выходе:

```
@Override
public void onDestroy() {
  super.onDestroy();

  // Закройте базу данных
  toDoDBAdapter.close();
}
```

Теперь задания из списка будут сохраняться между сессиями. Для дальнейшего улучшения вы можете заменить ArrayAdapter на SimpleCursorAdapter — это обеспечит динамическое обновление ListView при изменениях в базе данных.

Поскольку вы используете приватную базу данных, ваши задания не доступны для других приложений. Чтобы обеспечить доступ, нужно воспользоваться Источниками данных. Этим вы и займетесь в следующем разделе.

Создание нового Источника данных

Для создания нового Источника данных наследуйте абстрактный класс ContentProvider. Переопределите метод onCreate, чтобы создать (и инициализировать) базовый источник, который вы хотите опубликовать. Простой каркас для нового Источника данных показан в листинге 7.9.

Листинг 7.9. Создание нового Источника данных

```
import android.content.*;
import android.database.Cursor;
import android.net.Uri;
import android.database.SQLException;

public class MyProvider extends ContentProvider {

  @Override
  public boolean onCreate() {
    // TODO: Создать базу данных, с которой будет работать данный класс.
    return true;
  }
}
```

Нужно предусмотреть публичное статическое свойство CONTENT_URI, которое возвращает полный URI для данного источника. URI Источника данных должен быть уникальным, поэтому лучше привязать его к имени вашего пакета. В общем виде определение URI Источника данных выглядит так:

```
content://com.<CompanyName>.provider.<ApplicationName>/<DataPath>
```

Например:

```
content://com.paad.provider.myapp/elements
```

Пути URI могут представляться двумя способами. URI, показанный выше, — это запрос ко всем значениям определенного типа (в данным случае все элементы).

Чтобы получить запрос к конкретной строке, в конце нужно указать ее номер (в данном случае пятый элемент).

```
content://com.paad.provider.myapp/elements/5
```

Поддержка обоих видов доступа к вашему Источнику данных — правильное решение.

Самый простой способ обеспечить ее — использовать UriMatcher. Создайте и настройте UriMatcher, чтобы анализировать пути URI и распознавать их виды. Это чрезвычайно важно при обработке запросов из класса ContentResolver. Листинг 7.10 содержит каркас с кодом для подобной модели взаимодействия.

Листинг 7.10. Использование UriMatcher для обработки одиночных или множественных запросов к Источнику данных

```
public class MyProvider extends ContentProvider {

  private static final String myURI = "content://com.paad.provider.myapp/
items";
  public static final Uri CONTENT_URI = Uri.parse(myURI);

  @Override
  public boolean onCreate() {
    // TODO: Создать базу данных, с которой будет работать данный класс.
    return true;
  }

  // Создайте константы для распознавания разных запросов.
  private static final int ALLROWS = 1;
  private static final int SINGLE_ROW = 2;

  private static final UriMatcher uriMatcher;

  // Заполните данными объект UriMatcher. URI заканчивается на 'items',
  // когда осуществляется запрос ко всем элементам, и на 'items/[rowID]',
  // когда идет запрос к единственной строке.
  static {
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI("com.paad.provider.myApp", "items", ALLROWS);
    uriMatcher.addURI("com.paad.provider.myApp", "items/#", SINGLE_ROW);
  }
}
```

Вы можете использовать тот же подход при предоставлении альтернативных путей URI для разных наборов данных или разных таблиц в вашей базе данных с помощью одного и того же Источника данных.

Не помешает также назначить имена каждому столбцу, доступному в вашем Источнике данных, чтобы упростить извлечение данных из результирующего Курсора.

Предоставление доступа к Источнику данных

Упростите доступ к запросам и транзакциям в вашем Источнике данных, реализовав методы delete, insert, update и query.

Эти методы — интерфейс, используемый объектом ContentResolver для доступа к исходным данным. Они позволяют приложениям обмениваться данными в любой точке и не требуют разных интерфейсов для каждого источника данных.

Как правило, Источник данных применяется для работы с приватной базой данных SQLite, но, используя эти методы, вы можете получать доступ к любым источникам (включая файлы или поля, принадлежащие экземпляру приложения).

В листинге 7.11 описан каркас для реализации запросов и транзакций внутри Источника данных. Обратите внимание, что объект UriMatcher задействуют для уточнения этих запросов и транзакций.

Листинг 7.11. Реализация запросов и транзакций в Источнике данных

```
@Override
public Cursor query(Uri uri,
                    String[] projection,
                    String selection,
                    String[] selectionArgs,
                    String sort) {

  // Если это запрос для получения строки, ограничьте результат одной
  // строкой.
  switch (uriMatcher.match(uri)) {
    case SINGLE_ROW :
      // TODO: Изменять выборку, в зависимости от идентификатора строки,
      // где: rowNumber = uri.getPathSegments().get(1));
  }
  return null;
}

@Override
public Uri insert(Uri _uri, ContentValues _initialValues) {
  long rowID = [ ... Добавляем новый элемент ... ]

  // Верните путь URI к последнему добавленному элементу.
 if (rowID > 0) {
```

Продолжение ⇗

Листинг 7.11 (продолжение)

```
      return ContentUris.withAppendedId(CONTENT_URI, rowID);
    }
    throw new SQLException("Failed to add new item into " + _uri);
  }

  @Override
  public int delete(Uri uri, String where, String[] whereArgs) {
    switch (uriMatcher.match(uri)) {
      case ALLROWS:
      case SINGLE_ROW:
      default: throw new IllegalArgumentException("Unsupported URI:" + uri);
    }
  }

  @Override
  public int update(Uri uri, ContentValues values, String where, String[]
  whereArgs) {
    switch (uriMatcher.match(uri)) {
      case ALLROWS:
      case SINGLE_ROW:
      default: throw new IllegalArgumentException("Unsupported URI:" + uri);
    }
  }
```

Последний шаг при создании Источника данных — определение типа MIME, с помощью которого распознается тип данных, возвращенный Источником данных.

Переопределите метод getType, чтобы он возвращал уникальную строку, описывающую ваш тип данных. Возвращаемый тип должен включать две формы: одну для одиночной записи, а другую для всех записей сразу. Это будет выглядеть так:

- одиночная запись: `vnd.<companyname>.cursor.item/<contenttype>`;
- все записи: `vnd.<companyName>.cursor.dir/<contenttype>`.

В листинге 7.12 показывается, как переопределить метод getType, чтобы он возвращал корректный тип MIME, основываясь на переданном URI.

Листинг 7.12. Возвращение типа MIME Источника данных

```
  @Override
  public String getType(Uri _uri) {
    switch (uriMatcher.match(_uri)) {
      case ALLROWS: return "vnd.paad.cursor.dir/myprovidercontent";
      case SINGLE_ROW: return "vnd.paad.cursor.item/myprovidercontent";
      default: throw new IllegalArgumentException("Unsupported
  URI: " + _uri);
    }
  }
```

Регистрация Источника данных

После того как вы создали свой Источник данных, его нужно добавить в манифест приложения.

Используйте тег `authorities`, чтобы указать базовый путь URI, как показано в следующем фрагменте кода XML:

```
<provider android:name="MyProvider"
          android:authorities="com.paad.provider.myapp"/>
```

Использование Источников данных

В следующих разделах вы познакомитесь с классом ContentResolver, узнаете, как его использовать для запросов и транзакций совместно с Источником данных.

Знакомство с ContentResolver

Каждый объект Context, принадлежащий приложению, включает в себя экземпляр класса ContentResolver, доступ к которому обеспечивает метод getContentResolver.

```
ContentResolver cr = getContentResolver();
```

ContentResolver применяет методы для изменения данных и выполнения запросов к Источникам данных. Каждый метод принимает URI, который указывает на то, с каким именно Источником данных нужно работать.

Путь URI Источника данных определяет его полномочия, описанные в соответствующем разделе манифеста. URI может быть произвольной строкой, так что большинство Источников данных имеют публичное свойство CONTENT_URI, чтобы предоставлять этот путь другим объектам.

Источники данных, как правило, предоставляют два вида URI: один для запросов, требующих все данные сразу, а другой для запросов, которые возвращают одиночную строку. В последнем случае к CONTENT_URI в конце добавляется `/<rowID>`.

Запросы для получения данных

Запросы к Источникам данных весьма похожи на запросы к базам данных. Как и в случае с базами, результат приходит в виде результирующего объекта Cursor. Все работает по тому же принципу, который описан ранее в этой главе.

Вы можете извлекать значения из результирующего Курсора с помощью тех же методов, о которых шла речь в разделе «Извлечение результатов из объекта Cursor».

Используя метод query, принадлежащий объекту ContentResolver, нужно передать ему:

- путь URI Источника данных, к которому требуется выполнить запрос;
- проекцию, содержащую список столбцов, которые хотите включить в результирующий набор данных;
- оператор WHERE, определяющий строки для возвращения (вы можете использовать маски «?», их заменят переданные значения для оператора SELECT);
- массив строк, выступающих в роли аргументов для оператора SELECT, которые должны заменить маски «?» в операторе WHERE;
- строку, описывающую порядок возвращения результирующих строк.

В листинге 7.13 показано, как использовать ContentResolver для запроса к Источнику данных.

Листинг 7.13. Создание запросов к Источнику данных с помощью ContentResolver

```
ContentResolver cr = getContentResolver();
// Верните все строки
Cursor allRows = cr.query(MyProvider.CONTENT_URI, null, null, null, null);
// Верните все столбцы для строк, в которых третий столбец равен
// заданному значению.
// Строки отсортированы по пятому столбцу.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor someRows = cr.query(MyProvider.CONTENT_URI,
                        null, where, null, order);
```

Далее в этой главе, после того как познакомитесь со стандартными Источниками данных в Android, представлено больше примеров запросов для получения данных.

Добавление, обновление и удаление данных

Для передачи транзакций к Источникам данных, используйте методы delete, update и insert, принадлежащие объекту ContentResolver.

Вставка

Класс ContentResolver предлагает два метода вставки новых записей в Источник данных — insert и bulkInsert. Оба принимают в качестве пара-

метра путь URI, описывающий тип элементов, которые вы добавляете, но первый метод — одиночный объект ContentValues, а второй — массив этих объектов.

Обычный метод insert вернет путь URI к только что добавленной записи, а bulkInsert вернет количество записей, которые были успешно добавлены.

В листинге 7.14 показано, как использовать методы insert и bulkInsert.

Листинг 7.14. Вставка новых строк в Источник данных

```
// Получите объект ContentResolver
ContentResolver cr = getContentResolver();

// Создайте новую строку со значениями, которые вы хотите вставить.
ContentValues newValues = new ContentValues();

// Назначьте значения для каждой строки.
newValues.put(COLUMN_NAME, newValue);
[ ... Повторите для каждого столбца ... ]

Uri myRowUri = cr.insert(MyProvider.CONTENT_URI, newValues);

// Создайте новую строку со значениями, которые вы хотите вставить.
ContentValues[] valueArray = new ContentValues[5];

// TODO: Создать массив новых строк
int count = cr.bulkInsert(MyProvider.CONTENT_URI, valueArray);
```

Удаление

Чтобы удалить одиночную запись, вызовите метод delete из объекта ContentResolver, передав ему путь URI той строки, которую хотите убрать. Кроме этого можно указать оператор WHERE для удаления нескольких строк. Оба подхода показаны в листинге 7.15.

Листинг 7.15. Удаление записей из Источника данных

```
ContentResolver cr = getContentResolver();

// Удалите конкретную строку.
cr.delete(myRowUri, null, null);
// Удалите первых пять строк.
String where = "_id < 5";
cr.delete(MyProvider.CONTENT_URI, where, null);
```

Обновление

Строки в Источнике данных обновляются с помощью метода update, принадлежащего ContentResolver. Данный метод принимает в качестве параметров путь URI, ссылающийся на нужный Источник данных, объект ContentValues, связывающий имена столбцов с обновленными данными,

а также оператор WHERE, с помощью которого можно выбрать строки для обновления.

В процессе каждая строка, подходящая под описание оператора WHERE, обновляется, используя переданный объект ContentValues. При этом возвращается количество успешных обновлений, как показано в листинге 7.16.

Листинг 7.16. Обновление записей в Источнике данных

```
// Создайте новую строку со значениями, которые вы хотите вставить.
ContentValues newValues = new ContentValues();

// Создайте ассоциативный массив, который определяет, какой столбец
// вы хотите обновить, и какие значения ему будут присвоены.
newValues.put(COLUMN_NAME, newValue);

// Примените изменения к первым пяти строкам.
String where = "_id < 5";

getContentResolver().update(MyProvider.CONTENT_URI, newValues, where,
null);
```

Доступ к файлам в рамках Источников данных

Источники данных представляют файлы в виде полноценного пути URI, вместо «сырого» файлового объекта. Чтобы вставить файл в Источник данных или получить доступ к сохраненному, используйте методы openOutputStream и openInputStream соответственно. Процесс показан в листинге 7.17.

Листинг 7.17. Добавление файла в Источник данных

```
// Вставьте новую строку в ваш Источник, получите ее уникальный путь URI.
Uri uri = getContentResolver().insert(MyProvider.CONTENT_URI, newValues);

try {
  // Откройте исходящий поток, используя URI новой строки.
  OutputStream outStream = getContentResolver().openOutputStream(uri);
  // Сожмите изображение и сохраните его в Источнике данных.
  sourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outStream);
}
catch (FileNotFoundException e) { }
```

Создание и использование Источника данных для приложения Earthquake

Вы уже создали приложение, которое выводит список землетрясений. Теперь появилась отличная возможность поделиться этим списком с другими приложениями.

Предоставление этой информации с помощью Источника данных позволит вам и другим разработчикам создавать новые приложения, основанные на

данных о землетрясениях, без необходимости дублировать всю работу по передаче данных через сеть и по обработке XML.

Создание Источника данных[1]

1. Откройте проект Earthquake и создайте новый класс EarthquakeProvider, наследующий ContentProvider. Переопределите методы onCreate, getType, query, insert, delete и update с помощью заглушек.

```
package com.paad.earthquake;

import android.content.*;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
import android.util.Log;

public class EarthquakeProvider extends ContentProvider {

  @Override
  public boolean onCreate() {
  }

  @Override
  public String getType(Uri url) {
  }

  @Override
  public Cursor query(Uri url, String[] projection, String selection,
                      String[] selectionArgs, String sort) {
  }

  @Override
  public Uri insert(Uri _url, ContentValues _initialValues) {
  }

  @Override
  public int delete(Uri url, String where, String[] whereArgs) {
  }

  @Override
  public int update(Uri url, ContentValues values,
                    String where, String[]wArgs) {
  }
}
```

[1] Все фрагменты кода в этом примере — часть проекта To-Do List 2 из главы 7, их можно загрузить с сайта Wrox.com.

2. Сделайте доступным путь URI для этого источника. Он понадобится для доступа к Источнику данных из других компонентов приложения через объект ContentResolver.

```
public static final Uri CONTENT_URI =
  Uri.parse("content://com.paad.provider.earthquake/earthquakes");
```

3. Создайте базу данных, которая задействуется для хранения информации о землетрясениях. Внутри EarthquakeProvider сделайте новый экземпляр класса SQLiteDatabase и добавьте публичные поля, описывающие имена и индексы столбцов. Наследуйте класс SQLiteOpenHelper, чтобы управлять созданием базы данных и контролировать ее версии.

```
// Исходная база данных
private SQLiteDatabase earthquakeDB;

private static final String TAG = "EarthquakeProvider";
private static final String DATABASE_NAME = "earthquakes.db";
private static final int DATABASE_VERSION = 1;
private static final String EARTHQUAKE_TABLE = "earthquakes";

// Имена столбцов
public static final String KEY_ID = "_id";
public static final String KEY_DATE = "date";
public static final String KEY_DETAILS = "details";
public static final String KEY_LOCATION_LAT = "latitude";
public static final String KEY_LOCATION_LNG = "longitude";
public static final String KEY_MAGNITUDE = "magnitude";
public static final String KEY_LINK = "link";

// Индексы столбцов
public static final int DATE_COLUMN = 1;
public static final int DETAILS_COLUMN = 2;
public static final int LONGITUDE_COLUMN = 3;
public static final int LATITUDE_COLUMN = 4;
public static final int MAGNITUDE_COLUMN = 5;
public static final int LINK_COLUMN = 6;

// Вспомогательный класс для открытия, создания и управления контролем
версий базы данных
private static class earthquakeDatabaseHelper extends SQLiteOpenHelper {
  private static final String DATABASE_CREATE =
    "create table " + EARTHQUAKE_TABLE + " ("
    + KEY_ID + " integer primary key autoincrement, "
    + KEY_DATE + " INTEGER, " + KEY_DETAILS + " TEXT, "
    + KEY_LOCATION_LAT + " FLOAT, " + KEY_LOCATION_LNG + " FLOAT, "
    + KEY_MAGNITUDE + " FLOAT), "
    + KEY_LINK + " TEXT);";

  public earthquakeDatabaseHelper(Context context, String name,
                                  CursorFactory factory, int version) {
    super(context, name, factory, version);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
```

```
    db.execSQL(DATABASE_CREATE);
  }

  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");

    db.execSQL("DROP TABLE IF EXISTS " + EARTHQUAKE_TABLE);
    onCreate(db);
  }
}
```

4. Создайте объект UriMatcher для обработки запросов, которые используют разные URI. Добавьте поддержку запросов и транзакций, возвращающих полный набор данных (QUAKES) или одиночную запись по значению индекса (QUAKE_ID).

```
// Создайте константы для запросов к разным URI.
private static final int QUAKES = 1;
private static final int QUAKE_ID = 2;

private static final UriMatcher uriMatcher;

// Создайте объект UriMatcher, сделайте так, чтобы путь URI,
заканчивающийся на
// 'earthquakes', соответствовал запросу ко всем землетрясениям, а URI,
// имеющий окончание '/[rowID]', соответствовал одиночной строке с
// информацией о землетрясении.
static {
  uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
  uriMatcher.addURI("com.paad.provider.Earthquake", "earthquakes",
QUAKES);
  uriMatcher.addURI("com.paad.provider.Earthquake", "earthquakes/#",
QUAKE_ID);
}
```

5. Переопределите метод getType, чтобы возвращать строковое значение для каждого пути URI, который имеет поддерживаемую структуру.

```
@Override
public String getType(Uri uri) {
  switch (uriMatcher.match(uri)) {
    case QUAKES: return "vnd.android.cursor.dir/vnd.paad.earthquake";
    case QUAKE_ID: return "vnd.android.cursor.item/vnd.paad.earthquake";
    default: throw new IllegalArgumentException("Unsupported URI: " + uri);
  }
}
```

6. Переопределите обработчик onCreate, принадлежащий Источнику данных, чтобы создать новый экземпляр вспомогательного класса для работы с базой данных и открыть соединение к ней.

```
@Override
public boolean onCreate() {
```

```
  Context context = getContext();

  earthquakeDatabaseHelper dbHelper = new earthquakeDatabaseHelper(context,
    DATABASE_NAME, null, DATABASE_VERSION);
  earthquakeDB = dbHelper.getWritableDatabase();
  return (earthquakeDB == null) ? false : true;
}
```

7. Наполните кодом заглушки для транзакций и запросов. Начните с метода query, который должен расшифровывать поступивший запрос, основанный на URI (определять, требуется все содержимое или только одна строка), и применять параметры (выборку, проекцию и порядок сортировки) к базе данных, прежде чем возвращать результирующий объект Cursor.

```
@Override
public Cursor query(Uri uri,
                    String[] projection,
                    String selection,
                    String[] selectionArgs,
                    String sort) {

  SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

  qb.setTables(EARTHQUAKE_TABLE);

  // Если это запрос к одной строке, ограничьте результат требуемой строкой
  switch (uriMatcher.match(uri)) {
    case QUAKE_ID: qb.appendWhere(KEY_ID + "=" + uri.getPathSegments().
get(1));
                   break;
    default      : break;
  }

  // Если не указан порядок сортировки, сортируйте по дате/времени
  String orderBy;
  if (TextUtils.isEmpty(sort)) {
    orderBy = KEY_DATE;
  } else {
    orderBy = sort;
  }

  // Примените запрос к исходной базе данных.
  Cursor c = qb.query(earthquakeDB,
                      projection,
                      selection, selectionArgs,
                      null, null,
                      orderBy);

  // Зарегистрируйте ContentResolver, принадлежащий объекту Context, чтобы
  // получать уведомления об изменении результирующего набора данных.
  c.setNotificationUri(getContext().getContentResolver(), uri);

  // Верните курсор, ссылающийся на результат запроса.
  return c;
}
```

8. Реализуйте методы для вставки, удаления и обновления содержимого, относящегося к Источнику данных. В данном случае работа заключается в связывании транзакционных запросов к Источнику данных с их эквивалентами для базы данных.

```
@Override
public Uri insert(Uri _uri, ContentValues _initialValues) {
  // Вставьте новую строку, в случае успеха будет возвращен
  // номер строки.
  long rowID = earthquakeDB.insert(EARTHQUAKE_TABLE, "quake", _
initialValues);

  // Верните путь URI к только что вставленной строке
  // (в случае успеха).
  if (rowID > 0) {
    Uri uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
    getContext().getContentResolver().notifyChange(uri, null);
    return uri;
  }
  throw new SQLException("Failed to insert row into " + _uri);
}

@Override
public int delete(Uri uri, String where, String[] whereArgs) {
  int count;

  switch (uriMatcher.match(uri)) {
    case QUAKES:
      count = earthquakeDB.delete(EARTHQUAKE_TABLE, where, whereArgs);
      break;

    case QUAKE_ID:
      String segment = uri.getPathSegments().get(1);
      count = earthquakeDB.delete(EARTHQUAKE_TABLE, KEY_ID + "="
                                  + segment
                                  + (!TextUtils.isEmpty(where) ? " AND ("
                                  + where + ')' : ""), whereArgs);
      break;

    default: throw new IllegalArgumentException("Unsupported URI: " + uri);
  }

  getContext().getContentResolver().notifyChange(uri, null);
  return count;
}

@Override
public int update(Uri uri, ContentValues values, String where, String[]
whereArgs) {
  int count;
  switch (uriMatcher.match(uri)) {
    case QUAKES: count = earthquakeDB.update(EARTHQUAKE_TABLE, values,
      where, whereArgs);
      break;

    case QUAKE_ID: String segment = uri.getPathSegments().get(1);
```

```
                count = earthquakeDB.update(EARTHQUAKE_TABLE, values, KEY_ID
                          + "=" + segment
                          + (!TextUtils.isEmpty(where) ? " AND ("
                          + where + ')' : ""), whereArgs);
                break;

      default: throw new IllegalArgumentException("Unknown URI " + uri);
    }

    getContext().getContentResolver().notifyChange(uri, null);
    return count;
  }
```

9. Теперь, когда реализация Источника данных завершена, зарегистрируйте его в манифесте, создав новый узел внутри тега `application`.

```
<provider android:name=".EarthquakeProvider"
          android:authorities="com.paad.provider.earthquake" />
```

Использование собственного Источника данных[1]

Теперь вы можете обновить Активность приложения Earthquake, добавив в нее ранее созданный Источник данных, чтобы хранить землетрясения, которые понадобятся для заполнения элемента ListView.

1. Измените метод addNewQuake внутри Активности Earthquake. Он должен использовать объект ContentResolver для вставки каждой новой записи о землетрясении в Источник данных. Перенесите существующую логику по работе с массивом в отдельный метод addQuakeToArray.

```
private void addNewQuake(Quake _quake) {
  ContentResolver cr = getContentResolver();
  // Создайте оператор WHERE, чтобы быть уверенным, что Источник данных
  // уже не содержит это землетрясение.
  String w = EarthquakeProvider.KEY_DATE + " = " + _quake.getDate().
getTime();

  // Если землетрясение новое, вставьте его в Источник данных.
  if (cr.query(EarthquakeProvider.CONTENT_URI, null, w, null, null).
getCount()==0){
    ContentValues values = new ContentValues();

    values.put(EarthquakeProvider.KEY_DATE, _quake.getDate().getTime());
    values.put(EarthquakeProvider.KEY_DETAILS, _quake.getDetails());

    double lat = _quake.getLocation().getLatitude();
    double lng = _quake.getLocation().getLongitude();
    values.put(EarthquakeProvider.KEY_LOCATION_LAT, lat);
    values.put(EarthquakeProvider.KEY_LOCATION_LNG, lng);
    values.put(EarthquakeProvider.KEY_LINK, _quake.getLink());
```

[1] Все фрагменты кода в этом примере — часть проекта To-Do List 3 из главы 7, их можно загрузить с сайта Wrox.com.

```
    values.put(EarthquakeProvider.KEY_MAGNITUDE, _quake.getMagnitude());

    cr.insert(EarthquakeProvider.CONTENT_URI, values);
    earthquakes.add(_quake);

    addQuakeToArray(_quake);
  }
}

private void addQuakeToArray(Quake _quake) {
  if (_quake.getMagnitude() > minimumMagnitude) {
    // Добавьте новое землетрясение в наш список.
    earthquakes.add(_quake);

    // Оповестите Адаптер массива об изменениях.
    aa.notifyDataSetChanged();
  }
}
```

2. Создайте новый метод loadQuakesFromProvider, ответственный за загрузку всех землетрясений из Источника данных и вставку их в ArrayList с помощью метода addQuakeToArray из первого пункта.

```
private void loadQuakesFromProvider() {
  // Очистите существующий массив с землетрясениями
  earthquakes.clear();

  ContentResolver cr = getContentResolver();

  // Верните все сохраненные землетрясения
  Cursor c = cr.query(EarthquakeProvider.CONTENT_URI, null, null, null,
null);

  if (c.moveToFirst())
   {
    do {
      // Извлеките информацию о землетрясении.
      Long datems = c.getLong(EarthquakeProvider.DATE_COLUMN);
      String details = c.getString(EarthquakeProvider.DETAILS_COLUMN);
      Float lat = c.getFloat(EarthquakeProvider.LATITUDE_COLUMN);
      Float lng = c.getFloat(EarthquakeProvider.LONGITUDE_COLUMN);
      Double mag = c.getDouble(EarthquakeProvider.MAGNITUDE_COLUMN);
      String link = c.getString(EarthquakeProvider.LINK_COLUMN);

      Location location = new Location("dummy");
      location.setLongitude(lng);
      location.setLatitude(lat);

      Date date = new Date(datems);

      Quake q = new Quake(date, details, location, mag, link);
      addQuakeToArray(q);
    } while(c.moveToNext());
  }
}
```

3. Вызовите метод loadQuakesFromProvider из обработчика onCreate, чтобы инициализировать элемент ListView при запуске приложения.

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  setContentView(R.layout.main);

  earthquakeListView = (ListView)this.findViewById(R.
id.earthquakeListView);

  earthquakeListView.setOnItemClickListener(new OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView _av, View _v, int _index, long
arg3) {
      selectedQuake = earthquakes.get(_index);
      showDialog(QUAKE_DIALOG);
    }
  });

  int layoutID = android.R.layout.simple_list_item_1;
  aa = new ArrayAdapter<Quake>(this, layoutID , earthquakes);
  earthquakeListView.setAdapter(aa);

  loadQuakesFromProvider();

  updateFromPreferences();
  refreshEarthquakes();
}
```

4. Внесите изменения в метод refreshEarthquakes, чтобы он загружал сохраненные землетрясения из Источника данных после очистки массива, но перед добавлением новых землетрясений.

```
private void refreshEarthquakes() {
  [ ... ранее созданный метод refreshEarthquakes ... ]

  // Удалите информацию о старых землетрясениях
  earthquakes.clear();
  loadQuakesFromProvider();

  [ ... ранее созданный метод refreshEarthquakes ... ]
}
```

Стандартные Источники данных в Android

Android предоставляет доступ к нескольким стандартным базам данных, используя Источники данных.

Вы можете использовать эти Источники данных напрямую с помощью методов, описанных ранее в этой главе. Но у вас также есть возможность

воспользоваться пакетом android.provider, который содержит классы, упрощающие доступ ко многим из наиболее полезных источников.

- Browser. Используйте этот Источник данных для чтения или изменения закладок, истории посещений или использования поиска в обозревателе.
- CallLog. Выводит или обновляет историю звонков (входящие, исходящие, пропущенные), открывает доступ к детальной информации, такой как номер звонившего и продолжительность разговора.
- ContactsContract. Используйте этот Источник данных для получения, изменения или хранения информации о контактах. Он заменяет старый класс Contacts.
- MediaStore. Этот Источник данных предоставляет централизованный, управляемый доступ к файлам мультимедиа на вашем устройстве, включая аудио, видео и изображения. Вы можете хранить в нем собственные файлы мультимедиа и делать их общедоступными, как показано в главе 11.
- Settings. Вы можете получить доступ к настройкам устройства с помощью этого Источника данных. Предоставляется возможность просматривать большинство системных настроек, а некоторые из них даже изменять. Класс android.provider.Settings содержит коллекцию действий для Намерений, которые могут использоваться для открытия соответствующего экрана с настройками, чтобы пользователь мог их поменять.
- UserDictionary. Обеспечивает доступ к пользовательским наборам слов, добавленных в словарь для интеллектуального ввода текста.

Нужно использовать эти стандартные Источники данных везде, где возможно, чтобы обеспечить гармоничную интеграцию вашего приложения с другими программами (как системными, так и сторонними).

Хотя подробная информация о каждом из этих вспомогательных компонентов выходит за рамки данной главы, в следующих ниже разделах описывается принцип использования Источников данных MediaStore и ContactsContract.

Использование Источника данных MediaStore

MediaStore в Android можно назвать хранилищем аудио-, видеофайлов, а также изображений.

Каждый раз, когда вы записываете на файловую систему файл мультимедиа, он также должен быть добавлен в MediaStore. Это откроет доступ к нему для других приложений, включая стандартный медиапроигрыватель. В главе 11 вы узнаете, как использовать ContentScanner для добавления новых файлов в MediaStore.

Чтобы получить доступ к медиафайлам из MediaStore, нужно запрашивать их через Источники данных, как это описывалось ранее в данной главе. Класс MediaStore включает подклассы Audio, Video и Images, которые, в свою очередь, содержат подклассы, обеспечивающие доступ к именам столбцов и путям URI, ссылающимся на содержимое каждого Источника данных.

MediaStore упорядочивает файлы мультимедиа, хранящиеся на внутренних и внешних носителях устройства. Каждый из его подклассов предоставляет путь URI ко внутренним или внешним файлам, используя шаблоны вида:

- `MediaStore.<mediatype>.Media.EXTERNAL_CONTENT_URI`;
- `MediaStore.<mediatype>.Media.INTERNAL_CONTENT_URI`.

В листинге 7.18 показан фрагмент кода, в котором идет поиск названия песни и альбома в каждом аудиофайле, хранящемся на внутреннем или внешнем носителе.

Листинг 7.18. Доступ к Источнику данных MediaStore

```
// Получите курсор, ссылающийся на все аудиофайлы, хранящиеся
// на внутреннем или внешнем носителе.
Cursor cursor =
getContentResolver().query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
                                              null, null, null, null);

// Дайте возможность Активности управлять жизненным циклом курсора.
startManagingCursor(cursor);

// Используйте удобные свойства для получения индексов столбцов
int albumIdx = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.
ALBUM);
int titleIdx = cursor. getColumnIndexOrThrow(MediaStore.Audio.Media.
TITLE);

String[] result = new String[cursor.getCount()];
if (cursor.moveToFirst())
  do {
    // Извлеките название песни.
    String title = cursor.getString(titleIdx);
    // Извлеките название альбома.
    String album = cursor.getString(albumIdx);

    result[cursor.getPosition()] = title + " (" + album + ")";
  } while(cursor.moveToNext());
```

В главе 11 вы узнаете, как проигрывать аудио- и видеоресурсы, хранящиеся в MediaStore, указывая путь URI для определенного элемента мультимедиа.

Использование Источника данных ContactsContract

Доступ к управлению контактами понадобится, когда речь идет об устройствах для связи. Разработчики Android пошли верным путем и предоставляют всю информацию из базы данных контактов для любого приложения, имеющего полномочие READ_CONTACTS.

В Android 2.0 (API level 5) появился класс ContactsContract, заменивший Contacts, который раньше использовался для хранения и управления контактами на устройстве.

Этот Источник данных включает новые возможности по управлению контактами в Android, предоставляя базу данных со всей информацией, касающейся контактов. Все это позволяет пользователям указывать несколько Источников своих контактных данных. Но более важно то, что разработчики сами могут расширять информацию для каждого контакта или даже создавать альтернативные Источники для контактов и сопутствующих данных.

Знакомство с Источником данных ContactsContract

Источник данных ContactsContract — это расширяемая база данных, содержащая всю информацию, связанную с контактами.

Вместо использования одной строго определенной таблицы с контактной информацией ContactsContract оперирует трехуровневой моделью, отвечающей за хранение данных, связывание с контактом и объединение их для конкретного человека, используя следующие подклассы.

- Data. В исходной таблице каждая строка определяет набор личных данных (например, телефонные номера, адреса электронной почты и т. д.), разделенных типом MIME. Несмотря на предопределенный набор основных имен столбцов для каждого типа личных данных (доступных наряду с соответствующими MIME-типами, хранящимися в ContactsContract.CommonDataKinds), эта таблица может использоваться для хранения любого значения.

  То, какие именно данные находятся в конкретной строке, определяется с помощью типа MIME, указанного для нее. Универсальные столбцы способны хранить до 15 различных секций с данными разного типа.

  При добавлении новых данных в таблицу Data, нужно указать объект класса RawContacts для каждого связанного набора данных.

- RawContacts. Начиная с версии Android 2.0 пользователи могут вносить несколько контактных учетных записей (например, Gmail, Facebook и т. д.). Каждая строка в таблице RawContacts определяет учетную запись, с которой будут ассоциироваться данные из таблицы Data.

- Contacts. Эта таблица объединяет все строки из RawContacts, которые относятся к одному и тому же человеку.

На деле вы будете использовать таблицу Data для добавления, удаления или изменения данных, относящихся к существующим учетным записям, RawContacts — для создания и управления самими учетными записями, Contacts и Data — для получения доступа к базе данных и извлечения информации о контактах.

Чтение информации о контактах

Вы можете применять ContentResolver для запросов к любой из трех таблиц, описанных выше, используя статическую константу CONTENT_URI, предоставляемую каждым из этих классов. Все классы содержат статические свойства, описывающие имена столбцов исходной таблицы.

Чтобы получить доступ к любой контактной информации, необходимо добавить полномочие READ_CONTACTS к манифесту вашего приложения:

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

В листинге 7.19 показаны запросы к таблице Contacts для получения Курсора, ссылающегося на всех адресатов в адресной книге, в результате создается массив, содержащий имя каждого контакта и его уникальный идентификатор.

Листинг 7.19. Получение доступа к Источнику данных с контактами

```
// Получите курсор, ссылающийся на все собранные контакты.
Cursor cursor =
getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
                                         null, null, null, null);

// Дайте возможность Активности управлять жизненным циклом курсора.
startManagingCursor(cursor);

// Используйте удобные свойства для получения индексов столбцов
int nameIdx = cursor.getColumnIndexOrThrow(ContactsContract.Contacts.
DISPLAY_NAME);
int idIdx = cursor. getColumnIndexOrThrow(ContactsContract.Contacts._ID);

String[] result = new String[cursor.getCount()];
if (cursor.moveToFirst())
  do {
    // Извлеките имя.
    String name = cursor.getString(nameIdx);
    // Извлеките номер телефона.
    String id = cursor.getString(idIdx);

    result[cursor.getPosition()] = name + " (" + id + ")";
  } while(cursor.moveToNext());

stopManagingCursor(cursor);
```

Источник данных ContactsContract.Data используется для хранения всей информации о контактах — адресов, телефонных номеров и адресов электронной почты. Это лучший вариант для поиска подобных данных.

Таблица Data также применяется при поиске информации о заданном контакте. В большинстве случаев вы, вероятно, будете запрашивать данные о контакте, основываясь на отрывке его имени.

Для упрощения поисков Android предоставляет путь URI для запросов ContactsContract.Contacts.CONTENT_FILTER_URI. Добавьте к URI полное имя или его часть в качестве дополнительного сегмента пути. Для извлечения связанной с этим именем контактной информации найдите значение _ID в возвращенном объекте Cursor и используйте его для создания запроса к таблице Data.

Внутри таблицы Data содержание каждого столбца в строке зависит от типа MIME, указанного для этой строки. Поэтому каждый запрос к данной таблице должен фильтровать строки по типу MIME, чтобы явно извлекать данные.

В листинге 7.20 показано, как использовать имена столбцов с информацией о контактах, доступные в подклассах CommonDataKinds, чтобы извлечь из таблицы Data отображаемое имя и номер мобильного телефона конкретного контакта.

Листинг 7.20. Поиск информации о контакте после нахождения самого контакта

```
// Найдите контакт с помощью частичного совпадения в имени
Uri lookupUri =
Uri.withAppendedPath(ContactsContract.Contacts.CONTENT_FILTER_URI,
"kristy");

Cursor idCursor = getContentResolver().query(lookupUri, null, null, null,
                                             null);

String id = null;
if (idCursor.moveToFirst()) {
  int idIdx =
idCursor.getColumnIndexOrThrow(ContactsContract.Contacts._ID);
  id = idCursor.getString(idIdx);
}
idCursor.close();

if (id != null) {
  // Верните всю контактную информацию типа PHONE для найденного контакта
  String where = ContactsContract.Data.CONTACT_ID + " = " + id + " AND " +
                 ContactsContract.Data.MIMETYPE + " = '" +
                 ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE +
                 "'";

  Cursor dataCursor =
```

Продолжение ➩

Листинг 7.20 (продолжение)

```
      getContentResolver().query(ContactsContract.Data.CONTENT_URI,
      null, where, null, null);

    // Используйте удобные свойства для получения указателей столбцов
    int nameIdx = dataCursor.getColumnIndexOrThrow(ContactsContract.Data.
DISPLAY_NAME);
    int phoneIdx =

dataCursor.getColumnIndexOrThrow(ContactsContract.CommonDataKinds.Phone.
NUMBER);

    String[] result = new String[dataCursor.getCount()];
    if (dataCursor.moveToFirst())
      do {
        // Извлеките имя.
        String name = dataCursor.getString(nameIdx);
        // Извлеките номер телефона.
        String number = dataCursor.getString(phoneIdx);

        result[dataCursor.getPosition()] = name + " (" + number + ")";
      } while(dataCursor.moveToNext());
    dataCursor.close();
  }
```

Подкласс Contacts предоставляет путь URI для поиска телефонного номера, чтобы помочь найти контакт, связанный с номером телефона. Этот запрос оптимизирован для возвращения быстрых результатов об уведомлении о входящих звонках.

Используйте путь `ContactsContract.PhoneLookup.CONTENT_FILTER_URI`, добавляя к нему номер, чтобы найти дополнительные участки пути, как показано в листинге 7.21.

Листинг 7.21. Поиск телефонного номера во входящих звонках

```
String incomingNumber = "5551234";

Uri lookupUri =
Uri.withAppendedPath(ContactsContract.PhoneLookup.CONTENT_FILTER_URI,
                     incomingNumber);

Cursor idCursor = getContentResolver().query(lookupUri, null, null, null,
                                             null);

if (idCursor.moveToFirst()) {
  int nameIdx =
    idCursor.getColumnIndexOrThrow(ContactsContract.Contacts.DISPLAY_
NAME);
  String caller = idCursor.getString(nameIdx);
  Toast.makeText(getApplicationContext(), caller, Toast.LENGTH_LONG).
show();
}
idCursor.close();
```

В дополнение к статической контактной информации, рассмотренной выше, таблица ContactsContract.StatusUpdates содержит обновления статусов в социальных сетях и данные о доступности в системах мгновенных сообщений. Используя эту таблицу, вы можете искать или изменять статус или сообщение о присутствии для любого контакта, который привязан к учетной записи в социальных сетях и/или системах мгновенных сообщений.

Изменение и расширение контактной информации

Кроме создания запросов к базе данных контактов можно использовать эти Источники данных для изменения, удаления или вставки записей о контактах, добавив полномочие WRITE_CONTACTS в манифест вашего приложения.

Благодаря принципу расширяемости ContactsContract вы можете добавлять произвольные строки в таблице Data к любой учетной записи, которая хранится в виде RawContacts. На деле это не лучший способ расширения сторонних учетных записей, так как нельзя синхронизировать новую информацию с удаленным сервером.

Более удачное решение — создание собственного Адаптера для синхронизации, объединенного с другими сторонними данными об учетной записи. Процесс этот выходит за рамки данной книги, однако в целом возможно создать тип учетной записи контакта для собственного набора данных, добавив запись в Источник RawContacts.

Можно добавлять новые записи в Источник Data, которые будут привязаны к учетной записи контакта, созданной вами. После добавления ваши новые контактные данные объединятся с информацией, предоставляемой стандартными и сторонними Адаптерами, и станут доступны при запросах к Источнику ContactsContract, как описано в предыдущем разделе.

Резюме

В этой главе вы научились добавлять к своему приложению надежный слой для хранения данных, а также получать доступ к сторонним Источникам данных.

Android обеспечивает доступ к полнофункциональной системе управления реляционными базами данных для всех приложений. Эта небольшая, эффективная и надежная библиотека позволяет создавать реляционные базы для хранения данных приложения. Вы научились делиться приватными данными, особенно базами данных, с другими приложениями, используя Источники данных.

Все запросы к базам данных и к Источникам данных возвращают Курсор. Вы научились отправлять запросы и извлекать данные из результирующих объектов Cursor.

Кроме того, вы также узнали:

- как создавать новые базы данных SQLite;
- взаимодействовать с базами данных для вставки, обновления и удаления строк;
- использовать стандартные Источники данных, поставляемые с Android, чтобы получать доступ и управлять стандартными данными, такими как медиафайлы и контакты.

Можно сказать, что теперь вы овладели основами разработки для платформы Android. В остальных главах этой книги рассматриваются наиболее интересные дополнительные возможности Android.

В следующей главе вы познакомитесь с API для геолокации. Android предоставляет богатый набор геолокационных возможностей, включая сервисы, основанные на местоположении (такие как GPS), прямое и обратное геокодирование, а также полную интеграцию с картами Google (Google maps). Google maps позволяет создавать Активности, в основе которых — объекты карт, что может послужить предпосылкой для разработки собственных проектов.

Глава 8
КАРТЫ, ГЕОКОДИРОВАНИЕ И ГЕОЛОКАЦИОННЫЕ СЕРВИСЫ

Содержание главы

- Прямое и обратное геокодирование.
- Создание интерактивных карт с помощью MapView и MapActivity.
- Создание Наложений (Overlays) и добавление их на карту.
- Поиск местоположения с помощью геолокационых сервисов.
- Использование оповещений о близости нахождения.

Одна из характерных особенностей мобильных телефонов — портативность, поэтому нет ничего удивительного в том, что некоторые из главных достоинств платформы Android связаны с сервисами, которые позволяют находить, контекстуализировать и определять на карте физическое местоположение.

Можно создать Активности с картографическими возможностями, используя Google Maps в качестве элемента интерфейса. У вас полный доступ к карте, что позволяет контролировать настройки отображения, изменять уровень масштабирования и подстраивать дисплей. Используя Наложения, можно добавлять на карту аннотации и обрабатывать пользовательские нажатия, чтобы предоставлять контекстную картографическую информацию и обеспечивать функциональность.

В этой главе в том числе рассматриваются геолокационные сервисы (location-based services, LBS), которые позволяют находить текущее местоположение вашего устройства. Они включают такие технологии, как GPS и система определения координат с помощью сотовой связи, разработанная в Google. Вы можете указать, какую геолокационную технологию использовать (явно или косвенно), перечислив набор критериев, описывающих точность и другие параметры.

Карты и другие геолокационные сервисы используют широту и долготу для определения географического положения, но пользователи предпочитают

оперировать таким понятием, как адрес. Android предоставляет класс Geocoder, поддерживающий прямое и обратное геокодирование. С помощью Geocoder можно преобразовывать в обе стороны значения широты/долготы, а также реальные адреса.

Сочетание карт, геокодирования и геолокационных сервисов — мощный инструментарий, позволяющий мобильным приложениям использовать портативность телефона.

Использование геолокационных сервисов

Геолокационные сервисы (LBS) — обобщающий термин, описывающий различные технологии, которые применяются для поиска текущего местоположения устройства.

Обозначим две главные составляющие LBS:

- LocationManager — предоставляет интерфейс к геолокационным сервисам.
- LocationProvider — предоставляет разные Источники данных, основанные на различных технологиях определения местоположения.

С помощью LocationManager можно:

- получать текущее местоположение;
- отслеживать передвижение;
- настраивать оповещения о близости нахождения таким образом, чтобы они срабатывали при входе в указанную область и при выходе из нее;
- находить доступные Источники данных для определения местоположения.

Настройка эмулятора для тестирования геолокационных сервисов

Для определения текущего местоположения геолокационные сервисы взаимодействуют с аппаратным обеспечением устройства. Когда разрабатываете и тестируете приложение с помощью эмулятора, аппаратная начинка виртуализирована, а вы, вероятно, находитесь в одном и том же месте.

Чтобы справиться с ограничениями, Android предлагает обходные пути, с помощью которых можно эмулировать данные для LocationProvider — это

позволит тестировать приложения, использующие геолокацию. В этом разделе вы научитесь имитировать информацию из приемника GPS.

ПРИМЕЧАНИЕ

Если планируете разрабатывать с помощью эмулятора Android приложения, основанные на геолокации, то из этого раздела вы узнаете, как создавать среду выполнения, симулирующую реальное аппаратное обеспечение и изменения местоположения. Далее по главе предполагается, что вы используете примеры из данного раздела для обновления местоположения с помощью GPS_PROVIDER, запуская их в эмуляторе. Как вариант, можно использовать реальное устройство.

Изменение местоположения в эмуляторе с помощью LocationProvider

Используйте панель Location Controls, доступную из перспективы DDMS в Eclipse (рис. 8.1), чтобы задать изменения местоположения прямо в LocationProvider.

На рис. 8.1 показаны вкладки Manual и KML. С помощью вкладки Manual можете указать определенные долготу и широту. Вкладки KML и GPX позволяют загружать файлы KML (язык разметки Keyhole) и GPX (формат для хранения и обмена данными GPS) соответственно. Как только они загрузились, переходите к произвольным координатам или воспроизводите их последовательно.

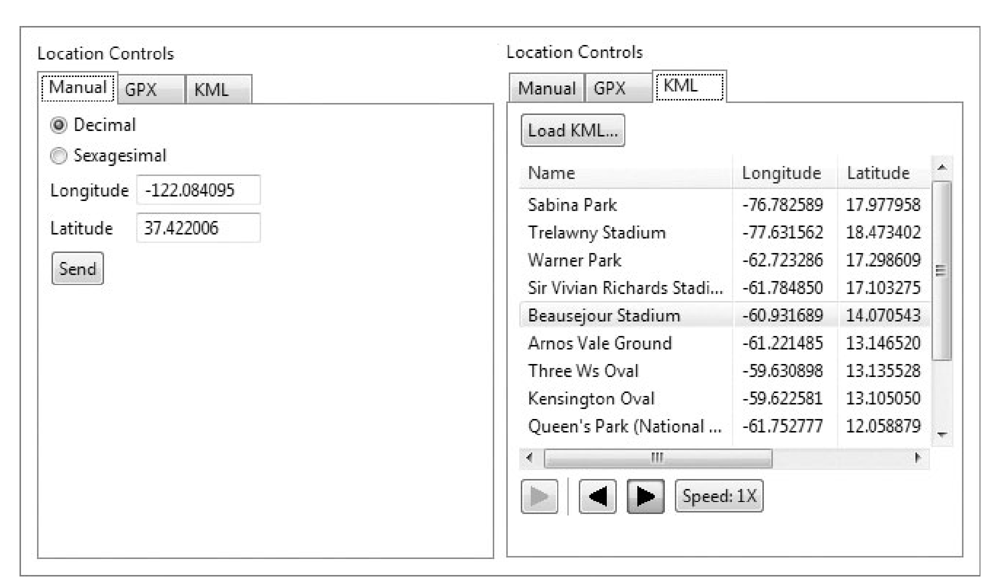


Рис. 8.1.

ПРИМЕЧАНИЕ

Большинство систем GPS записывают файлы с полученной информацией, используя GPX, тогда как KML широко распространен в Интернете и предназначен для определения географической информации. Вы можете вручную создать собственный файл KML или сгенерировать его с помощью приложения Google Earth, чтобы установить маршрут между двумя точками.

Все изменения местоположения, проведенные с помощью DDMS Location Controls, подхватываются приемником GPS, который должен быть включен и активизирован.

ВНИМАНИЕ

Данные GPS, возвращаемые методом getLastKnownLocation, не изменятся, пока хотя бы одно приложение не запросит обновление местоположения.

Выбор Источника данных для получения местоположения

В зависимости от устройства Android может применять несколько разных технологий для определения текущего местоположения. Каждая технология, или LocationProvider, предоставляет разные возможности (касается разницы в уровне энергопотребления, стоимости услуг и возможности определять высоту над уровнем моря, скорость или информацию о курсе).

Чтобы получить экземпляр конкретного Источника данных, вызовите метод getProvider, передав ему нужное название:

```
String providerName = LocationManager.GPS_PROVIDER;
LocationProvider gpsProvider;
gpsProvider = locationManager.getProvider(providerName);
```

Как правило, это нужно исключительно для определения возможностей конкретного источника. Большинство методов, принадлежащих классу LocationManager, требуют только имя источника, чтобы иметь возможность обращаться к геолокационным сервисам.

Поиск доступных Источников данных

Класс LocationManager содержит статические строковые константы, которые возвращают имя одного из двух базовых источников:

- `LocationManager.GPS_PROVIDER`;
- `LocationManager.NETWORK_PROVIDER`.

Чтобы получить список имен всех доступных источников на устройстве, вызовите метод getProviders, используя параметр типа Boolean (в зависимости от его значения вы получите или все источники сразу, или только те, которые сейчас активны):

```
boolean enabledOnly = true;
List<String> providers = locationManager.getProviders(enabledOnly);
```

Поиск Источников данных для получения местоположения с помощью класса Criteria

В большинстве случаев вы, вероятно, предпочтете не указывать явно тип LocationProvider. Чаще всего будете обозначать требования, которым должен отвечать источник, а Android сам определит наиболее подходящую технологию.

Используйте класс Criteria, чтобы перечислить требования к Источнику данных, касающиеся точности (высокая или низкая), уровня энергопотребления (низкий, средний или высокий), стоимости и возможности возвращать параметры высоты над уровнем моря, скорости и маршрута.

В листинге 8.1 с помощью Criteria указываются низкая точность, низкий уровень энергопотребления и отсутствие надобности в информации о высоте над уровнем моря, направлении или скорости. При этом могут тратиться соответствующие денежные средства.

Листинг 8.1. Создание Criteria для LocationProvider

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_COARSE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setSpeedRequired(false);
criteria.setCostAllowed(true);
```

Определив нужные параметры для Criteria, можете использовать метод getBestProvider, который возвращает наиболее подходящий тип LocationProvider, или метод getProviders, чтобы получить список источников, соответствующих требованиям. Следующий фрагмент показывает, как использовать getBestProvider для определения лучшего источника согласно описанным критериям, где параметр типа Boolean позволяет ограничиться только активизированными источниками:

```
String bestProvider = locationManager.getBestProvider(criteria, true);
```

Если указанным критериям соответствуют сразу несколько источников, возвратится тот, у которого больше всего совпадений. Если ни один из источников не подходит, объект Criteria начинает по очереди игнорировать

следующие параметры (в представленном порядке), пока источник не будет найден:

- энергопотребление;
- точность;
- возможность возвращать маршрут, скорость и высоту над уровнем моря.

Критерий, касающийся денежных затрат, никогда не игнорируется по умолчанию. Если не найдено ни одного источника, возвращается null.

Чтобы получить список имен всех источников, подпадающих под ваши требования, можете использовать метод getProviders. Он принимает объект Criteria и возвращает отфильтрованный список строк, который содержит все доступные типы LocationProvider, удовлетворяющие заданным критериям. Как и в случае с getBestProvider, если ни один из источников не будет найден, возвращается null.

```
List<String> matchingProviders = locationManager.getProviders(criteria,
                                                              false);
```

Поиск вашего местоположения

Цель геолокационных сервисов — определение физического местоположения вашего устройства.

За доступ к геолокационным сервисам отвечает системная служба LocationManager. Чтобы начать работу с ней, получите экземпляр типа LOCATION_SERVICE с помощью метода getSystemService, как показано в следующем фрагменте кода:

```
String serviceString = Context.LOCATION_SERVICE;
LocationManager locationManager;
locationManager = (LocationManager)getSystemService(serviceString);
```

Прежде чем использовать LocationManager, необходимо добавить один или несколько тегов, описывающих пользовательские полномочия (uses-permission), к манифесту приложения, чтобы получить доступ к аппаратному обеспечению, отвечающему за геолокационные сервисы.

В следующем фрагменте добавляются полномочия для высокой и низкой точности. Приложение, имеющее полномочия для высокой точности, автоматически получает доступ и к информации низкой точности.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

ПРИМЕЧАНИЕ

Источник GPS требует полномочий для использования данных высокой точности, тогда как сетевые источники (телефонные сети, Wi-Fi) могут обходиться и низкой точностью.

Вы можете получить последнее местоположение, зафиксированное LocationProvider, используя метод getLastKnownLocation, но предварительно передав ему в качестве параметра имя Источника данных. В следующем примере получено последнее зафиксированное местоположение, взятое у источника GPS:

```
String provider = LocationManager.GPS_PROVIDER;
Location location = locationManager.getLastKnownLocation(provider);
```

ПРИМЕЧАНИЕ

Метод getLastKnownLocation не инициирует обновление текущего местоположения у LocationProvider. Если устройство долго не обновляло текущую позицию, полученное значение может не существовать или быть неактуальным.

Возвращенный объект Location содержит всю позиционную информацию, которую поддерживает источник. Он может включать широту, долготу, азимут, высоту над уровнем моря, скорость и время, когда последний раз фиксировалось местоположение. Все эти свойства доступны через геттеры объекта Location. В некоторых случаях представлена дополнительная информация, содержащаяся в свойстве Bundle.

Приложение Where Am I?

Данный пример[1] — приложение Where Am I? — имеет новую Активность, которая ищет текущее местоположение устройства с помощью LocationProvider, использующего данные GPS. Далее в этой главе вы улучшите приведенный пример, когда изучите новые географические возможности.

ПРИМЕЧАНИЕ

Чтобы пример работал, нужно включить LocationProvider типа GPS_PROVIDER, как это показано ранее, или запустить его на устройстве, которое поддерживает GPS (при условии, что он включен).

1. Создайте новый проект Where Am I? с Активностью WhereAmI. В этом примере используется источник GPS (неважно, виртуальный или реальный), поэтому нужно отредактировать файл с манифестом, включив

[1] Все фрагменты кода в этом примере — часть проекта Where Am I? из главы 8, их можно загрузить с сайта Wrox.com.

в него тег <uses-permission> с именем ACCESS_FINE_LOCATION и INTERNET.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.paad.whereami">
  <application
    android:icon="@drawable/icon">
    <activity
      android:name=".WhereAmI"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"
  />
</manifest>
```

2. Отредактируйте файл main.xml, содержащий ресурс разметки, включив атрибут android:ID в тег элемента TextView, чтобы получить доступ к нему из Активности.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:id="@+id/myLocationText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
  />
  <uses permission
    android:name="android.permission.INTERNET
  />
</LinearLayout>
```

3. Переопределите метод onCreate, принадлежащий Активности WhereAmI, для того чтобы получить ссылку на LocationManager. Вызовите метод getLastKnownLocation, чтобы извлечь последнее зафиксированное значение местоположения, и передайте результат в качестве параметра для заглушки updateWithNewLocation.

```
package com.paad.whereami;

import android.app.Activity;
import android.content.Context;
```

```
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class WhereAmI extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    LocationManager locationManager;
    String context = Context.LOCATION_SERVICE;
    locationManager = (LocationManager)getSystemService(context);

    String provider = LocationManager.GPS_PROVIDER;
    Location location =
      locationManager.getLastKnownLocation(provider);

    updateWithNewLocation(location);
  }

  private void updateWithNewLocation(Location location) {}
}
```

4. Наполните кодом заглушку updateWithNewLocation. Этот метод должен выводить полученное местоположение с помощью элемента TextView, извлекая значения широты и долготы.

```
private void updateWithNewLocation(Location location) {
  String latLongString;
  TextView myLocationText;
  myLocationText = (TextView)findViewById(R.id.myLocationText);
  if (location != null) {
    double lat = location.getLatitude();
    double lng = location.getLongitude();
    latLongString = "Lat:" + lat + "\nLong:" + lng;
  } else {
    latLongString = "No location found";
  }
  myLocationText.setText("Your Current Position is:\n" +
                         latLongString);
}
```

При запуске Активность должна выглядеть, как на рис. 8.2.

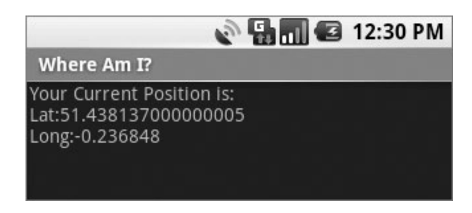


Рис. 8.2.

Отслеживание передвижений

Большинство приложений, имеющих дело с геолокацией, должны реагировать на передвижения пользователя. Простое запрашивание результатов у LocationManager не приносит новые обновления из LocationProvider.

Используйте метод requestLocationUpdates в связке с LocationListener для получения обновлений каждый раз, когда местоположение меняется. Объект LocationListener также генерирует события при изменении статуса и доступности источника.

Метод requestLocationUpdates принимает либо название конкретного типа LocationProvider, либо объект Criteria, содержащий список параметров для определения подходящего источника.

Чтобы увеличить эффективность, минимизируя при этом затраты и энергопотребление, вы также можете указать минимальные интервал и дистанцию между обновлениями данных о местоположении.

В листинге 8.2 показан каркас вызова регулярных обновлений, учитывающих минимальные интервал и дистанцию.

Листинг 8.2. Запрос на обновление данных о местоположении

```
String provider = LocationManager.GPS_PROVIDER;

int t = 5000; // миллисекунды
int distance = 5; // meters

LocationListener myLocationListener = new LocationListener() {

  public void onLocationChanged(Location location) {
    // Обновите приложение, основываясь на данных местоположения.
  }

  public void onProviderDisabled(String provider){
    // Обновите приложение, если источник отключен.
  }

  public void onProviderEnabled(String provider){
    // Обновите приложение, если источник включен.
  }

  public void onStatusChanged(String provider, int status,
                              Bundle extras){
    // Обновите приложение, если состояние аппаратного обеспечения
источника изменилось.
  }
};

locationManager.requestLocationUpdates(provider, t, distance,
                                       myLocationListener);
```

Когда значения минимальных интервала и дистанции превысятся, LocationListener вызовет собственное событие onLocationChanged.

ПРИМЕЧАНИЕ

Вы можете запрашивать сразу несколько обновлений для разных экземпляров LocationListener и использовать разные минимальные значения. Общий шаблон проектирования заключается в создании одиночного приемника событий для вашего приложения, который будет транслировать Намерения, оповещая другие компоненты об изменениях местоположения. Это централизует работу ваших приемников событий и гарантирует, что аппаратное обеспечение, связанное с LocationProvider, будет использовано максимально эффективно.

Чтобы остановить обновления информации о местоположении, вызовите метод removeUpdates, как показано в следующем фрагменте кода. Передайте ему экземпляр LocationListener, уведомления от которого вы больше не хотите получать.

```
locationManager.removeUpdates(myLocationListener);
```

Большинству приемников GPS требуются значительные энергозатраты. Чтобы минимизировать их, вы должны отключать обновления в вашем приложении каждый раз, когда это только возможно, особенно если ваше приложение не развернуто на экран, а изменения местоположения нужны для обновления пользовательского интерфейса Активности. Вы можете еще улучшить производительность, сделав минимальный интервал между обновлениями как можно более длинным.

Конфиденциальность важна, когда ваше приложение отслеживает местоположение пользователя. Убедитесь, что оно использует геолокационные данные устройства, а вы не забывайте с уважением относится к частной жизни пользователя:

- отслеживайте местоположение только тогда, когда это необходимо для вашего приложения;
- оповещайте пользователя о том, что начинаете отслеживать его перемещения, ознакомьте его с тем, как именно планируется хранить и использовать эту информацию;
- разрешите пользователю отключать обновления местоположения, учитывайте системные настройки для геолокационных сервисов.

Изменение вашего местоположения в программе Where Am I?

В следующем примере[1] приложение Where Am I? будет отслеживать ваше текущее местоположение, принимая сообщения о его изменении. Интервал обновлений — две секунды, и только при перемещении устройства более чем на 10 метров.

[1] Все фрагменты кода в этом примере — часть проекта Where Am I? из главы 8, их можно загрузить с сайта Wrox.com.

Вместо явного выбора источника GPS в этом примере вы создадите набор критериев и позволите Android самому выбрать наиболее подходящий доступный источник.

1. Сперва откройте Активность WhereAmI в проекте Where Am I?. Дополните метод onCreate, чтобы найти лучший вид LocationProvider, который имеет высокую точность и потребляет как можно меньше энергии.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  LocationManager locationManager;
  String context = Context.LOCATION_SERVICE;
  locationManager = (LocationManager)getSystemService(context);

  Criteria criteria = new Criteria();
  criteria.setAccuracy(Criteria.ACCURACY_FINE);

  criteria.setAltitudeRequired(false);
  criteria.setBearingRequired(false);
  criteria.setCostAllowed(true);
  criteria.setPowerRequirement(Criteria.POWER_LOW);
  String provider = locationManager.getBestProvider(criteria, true);

  Location location = locationManager.getLastKnownLocation(provider);
  updateWithNewLocation(location);
}
```

2. Создайте новый экземпляр класса LocationListener, который вызывает метод updateWithNew, принадлежащий объекту Location, каждый раз, когда фиксируется изменение местоположения.

```
private final LocationListener locationListener = new LocationListener() {
  public void onLocationChanged(Location location) {
    updateWithNewLocation(location);
  }

  public void onProviderDisabled(String provider){
    updateWithNewLocation(null);
  }

  public void onProviderEnabled(String provider){ }
  public void onStatusChanged(String provider, int status,
                              Bundle extras){ }
};
```

3. Вернитесь к обработчику onCreate и вызовите метод requestLocationUpdates, передав ему в качестве параметра новый объект LocationListener. Он должен следить за изменениями местоположения каждые

две секунды, но сигнализировать только тогда, когда устройство переместится более чем на 10 метров.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  LocationManager locationManager;
  String context = Context.LOCATION_SERVICE;
  locationManager = (LocationManager)getSystemService(context);

  Criteria criteria = new Criteria();
  criteria.setAccuracy(Criteria.ACCURACY_FINE);
  criteria.setAltitudeRequired(false);
  criteria.setBearingRequired(false);
  criteria.setCostAllowed(true);
  criteria.setPowerRequirement(Criteria.POWER_LOW);
  String provider = locationManager.getBestProvider(criteria, true);

  Location location =
    locationManager.getLastKnownLocation(provider);
  updateWithNewLocation(location);

  locationManager.requestLocationUpdates(provider, 2000, 10,
                                         locationListener);
}
```

Если вы запустите приложение и поменяете местоположение устройства, то увидите, что при этом изменяются данные в элементе TextView.

Использование оповещений о близости нахождения

Часто нужно сделать так, чтобы ваше приложение реагировало на приближение пользователя к какой-то точке или на его удаление от нее. Оповещения о близости нахождения позволяют приложению устанавливать триггеры, которые срабатывают в момент, когда пользователь приближается на определенное расстояние до какого-то географического местоположения или удаляется от него.

ПРИМЕЧАНИЕ

Внутри Android могут использоваться различные Источники данных для определения местоположения, в зависимости от того, как близко вы находитесь от границы целевой области. Это позволяет свести к минимуму денежные затраты и потребление энергии, так как оповещения будут генерироваться на основе данных о расстоянии от вас до внешней границы целевой области.

Чтобы установить оповещения о близости нахождения к данной области, нужно выбрать центральную точку (используя параметры широты и долготы), радиус вокруг этой точки и время ожидания для генерации оповещения. Оповещение вызовется, если устройство пересекло заданную границу, неважно, в каком направлении.

При срабатывании оповещения о близости нахождения генерируют Намерение, чаще всего это Широковещательное намерение. Чтобы указать, какое именно Намерение необходимо генерировать, используйте класс PendingIntent, который описывает Намерение в виде ссылки на метод, как показано во фрагменте кода:

```
Intent intent = new Intent(MY_ACTION);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, -1,
intent, 0);
```

В следующем примере устанавливается оповещение о близости нахождения, которое всегда остается актуальным и срабатывает в момент приближения устройства к цели ближе чем на 10 метров:

```
private static String TREASURE_PROXIMITY_ALERT = "com.paad.
treasurealert";

private void setProximityAlert() {
  String locService = Context.LOCATION_SERVICE;
  LocationManager locationManager;
  locationManager = (LocationManager)getSystemService(locService);

  double lat = 73.147536;
  double lng = 0.510638;
  float radius = 100f; // метры
  long expiration = -1; // время действия не истекает

  Intent intent = new Intent(TREASURE_PROXIMITY_ALERT);
  PendingIntent proximityIntent = PendingIntent.getBroadcast(this, -1,
                                                           intent,
                                                           0);
  locationManager.addProximityAlert(lat, lng, radius,
                                    expiration,
                                    proximityIntent);
}
```

Когда LocationManager обнаруживает, что вы пересекли границу радиуса (то есть переместились внутрь указанного радиуса близости или наоборот), упакованный объект Intent сработает с дополнительными ключами в зависимости от того, какое значение у LocationManager.KEY_PROXIMITY_ENTERING — true или false.

Для обработки оповещений о близости нахождения нужно создать объект BroadcastReceiver, как показано в листинге 8.3.

Листинг 8.3. Создание объекта BroadcastReceiver для приема оповещений о близости нахождения

```
public class ProximityIntentReceiver extends BroadcastReceiver {

  @Override
  public void onReceive (Context context, Intent intent) {
    String key = LocationManager.KEY_PROXIMITY_ENTERING;

    Boolean entering = intent.getBooleanExtra(key, false);
    [ . . . выполнение действий, связанных с обнаружением близости
нахождения .]
  }

}
```

Чтобы начать отслеживать оповещения о близости нахождения, зарегистрируйте приемник:

```
IntentFilter filter = new IntentFilter(TREASURE_PROXIMITY_ALERT);
registerReceiver(new ProximityIntentReceiver(), filter);
```

Использование геокодировщика

Геокодирование позволяет переводить уличные адреса в координаты широты/долготы и наоборот. Это предоставляет вам узнаваемый контекст для местоположения, а также координаты, используемые в геолокационных сервисах и Активностях, работающих с географическими картами.

Операции по геокодированию производятся на сервере, так что ваше приложение будет нуждаться в полномочиях, позволяющих использовать Интернет. Эти полномочия описываются в манифесте, как показано ниже:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Класс Geocoder обеспечивает доступ к двум функциям геокодирования:

- прямое геокодирование — находит координаты широты и долготы для указанного адреса;
- обратное геокодирование — находит уличный адрес для указанных широты и долготы.

Результаты вызовов этих функций зависят от региональных настроек (используются для определения вашего обычного местоположения и языка). В следующем фрагменте показано, как передать региональные настройки

при создании объекта Geocoder. Если вы этого не сделали, используются настройки по умолчанию.

```
Geocoder geocoder = new Geocoder(getApplicationContext(),
                                 Locale.getDefault());
```

Обе функции для геокодирования возвращают список с объектами Address. Каждый список может содержать несколько из возможных результатов, вплоть до максимального количества, указанного вами при вызове.

Каждый объект Address заполняется всеми данными, которые Geocoder смог получить. Сюда могут входить широта, долгота, телефонный номер, а также более детализированная информация: от страны до улицы и номера дома.

ПРИМЕЧАНИЕ

Геокодировщик производит поиск в синхронном режиме, поэтому он блокирует вызывающий его поток. Для медленных соединений это может привести к появлению диалога о принудительном закрытии приложения. В большинстве случаев хорошим решением будет переместить эти вызовы в отдельный Сервис или фононовый поток, как показано в главе 9.

Для краткости и ясности вызовы, которые содержатся в примерах кода из этой главы, осуществляются из главного потока приложения.

Обратное геокодирование

При обратном геокодировании возвращаются адреса, соответствующие физическому положению, указанному в виде пары широта — долгота. Это обеспечивает удобный для восприятия контекст, связанный с местоположением, возвращенным геолокационным сервисом.

Для обратного поиска нужно передать целевые широту и долготу в метод getFromLocation, принадлежащий объекту Geocoder. Он вернет список адресов, которые, вероятно, совпадают с переданными координатами. Если Geocoder не смог получить ни одного адреса для указанных координат, он вернет null.

В листинге 8.4 показывается, как применить обратное геокодирование к вашему последнему известному местоположению.

Листинг 8.4. Обратное геокодирование вашего последнего известного местоположения

```
location =
  locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

double latitude = location.getLatitude();
```

```
double longitude = location.getLongitude();
List<Address> addresses = null;

Geocoder gc = new Geocoder(this, Locale.getDefault());
try {
  addresses = gc.getFromLocation(latitude, longitude, 10);
} catch (IOException e) {}
```

Точность и детализация обратного геокодирования зависят от качества информации в базах данных геокодировщика, поэтому качество результатов может сильно варьироваться в зависимости от страны и выбранного языка.

Прямое геокодирование

Прямое геокодирование определяет географические координаты заданного местоположения.

ПРИМЕЧАНИЕ

Определение «правильное местоположение» варьируется в зависимости от места (географической области), где выполняется поиск. Как правило, оно будет включать стандартные адреса разной степени детализации (от страны до названия улицы и номера дома), почтовые индексы, железнодорожные станции, достопримечательности и больницы. По большому счету, правильные условия поиска похожи на адреса и места, которые можно ввести в строку поиска на сайте Google Maps.

Для прямого геокодирования вызовите метод getFromLocationName из экземпляра класса Geocoder. Передайте ему местоположение, координаты которого хотите получить, а также максимальное число допустимых результатов:

```
List<Address> result = geocoder.getFromLocationName(aStreetAddress,
maxResults);
```

Возвращенный список объектов Address может содержать несколько совпадений для заданного местоположения. Каждый объект будет включать широту, долготу и любую другую дополнительную информацию, доступную для этих координат. Это может пригодиться при подтверждении корректности полученного местоположения и указании особенностей местности при поиске достопримечательностей и ориентиров.

ПРИМЕЧАНИЕ

Как и в случае с обратным геокодированием, если не найдено ни одного совпадения, будет возвращен null. Доступность, точность и детализация результатов геокодирования полностью зависят от базы данных, содержащей информацию для местности, из которой ведется поиск.

Важную роль при прямом геокодировании играет объект Locale, переданный геокодировщику. Locale предоставляет географический контекст для интерпретации поисковых запросов, потому как одни и те же названия местностей могут находиться в разных местах. По возможности выбирайте Locale для конкретного региона, чтобы избежать неоднозначности в географических названиях.

Кроме того, пытайтесь использовать как можно больше информации об адресе. Пример показан в листинге 8.5.

Листинг 8.5. Геокодирование адреса

```
Geocoder fwdGeocoder = new Geocoder(this, Locale.US);
String streetAddress = "160 Riverside Drive, New York, New York";

List<Address> locations = null;
try {
  locations = fwdGeocoder.getFromLocationName(streetAddress, 10);
} catch (IOException e) {}
```

Для более конкретных результатов задействуйте перегруженный метод getFromLocationName, который позволяет ограничить поиск географическими рамками.

```
List<Address> locations = null;
try {
  locations = fwdGeocoder.getFromLocationName(streetAddress, 10,
                                              n, e, s, w);
} catch (IOException e) {}
```

Данный перегруженный метод эффективен в сочетании с элементом управления MapView, так как вы можете ограничить поиск районом, отображаемым на карте.

Геокодирование в приложении Where Am I?

С помощью геокодировщика можно определять адрес текущего местоположения. В этом примере[1] вы продолжите дополнять проект Where Am I? — внесете в него получение и обновление текущего адреса при перемещении устройства.

Начните с редактирования манифеста, добавив полномочия на использования Интернета:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Следующий шаг — откройте Активность WhereAmI. Отредактируйте метод updateWithNewLocation, чтобы инициализировать новый объект

[1] Все фрагменты кода в этом примере — часть проекта Where Am I? из главы 8, их можно загрузить с сайта Wrox.com.

Geocoder и вызвать метод getFromLocation, передав ему только что полученное местоположение и ограничив количество адресов до одного.

Извлеките все строки в адресе, а также район, почтовый индекс и страну, добавьте эту информацию к ранее созданной строке в TextView.

```
private void updateWithNewLocation(Location location) {
  String latLongString;
  TextView myLocationText;
  myLocationText = (TextView)findViewById(R.id.myLocationText);

  String addressString = "No address found";

  if (location != null) {
    double lat = location.getLatitude();
    double lng = location.getLongitude();
    latLongString = "Lat:" + lat + "\nLong:" + lng;

    double latitude = location.getLatitude();
    double longitude = location.getLongitude();
    Geocoder gc = new Geocoder(this, Locale.getDefault());
    try {
      List<Address> addresses = gc.getFromLocation(latitude, longitude,
1);
      StringBuilder sb = new StringBuilder();
      if (addresses.size() > 0) {
        Address address = addresses.get(0);

        for (int i = 0; i < address.getMaxAddressLineIndex(); i++) {
          sb.append(address.getAddressLine(i)).append("\n");

          sb.append(address.getLocality()).append("\n");
          sb.append(address.getPostalCode()).append("\n");
          sb.append(address.getCountryName());
        }
      }
      addressString = sb.toString();
    } catch (IOException e) {}
  } else {
    latLongString = "No location found";
  }
  myLocationText.setText("Your Current Position is:\n" +
                         latLongString + "\n" + addressString);
}
```

При запуске приложение должно выглядеть, как на рис. 8.3.

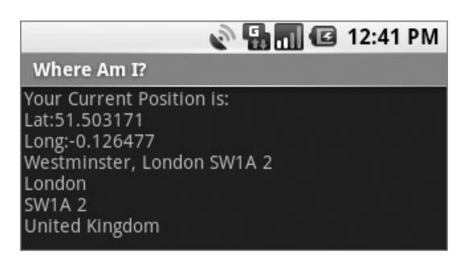


Рис. 8.3.

Создание Активностей, основанных на MapView

Элемент MapView предлагает идеальный вариант пользовательского интерфейса для отображения географических данных.

Один из наиболее интуитивных способов предоставить контекст для физического местоположения — отобразить его на карте. MapView позволяет создавать Активности, обладающие возможностями интерактивной карты.

MapView поддерживает нанесение аннотаций с помощью Наложений и закрепление Представлений за определенной точкой на карте. Класс MapView предоставляет полный контроль над внешним видом карты, позволяет менять масштаб, местоположение и режим отображения, включая спутниковый, уличный и транспортный.

Далее вы узнаете, как применить Наложения и MapController для создания динамических Активностей, использующих картографию. В отличие от онлайн-проектов ваши Активности будут работать непосредственно на устройстве, что позволит задействовать аппаратные средства и улучшить мобильность, а это в итоге сделает работу вашего приложения более гибкой и персонифицированной.

Знакомство с MapView и MapActivity

В этом разделе вы познакомитесь с несколькими классами, которые используются для поддержки картографии в Android.

- MapView. Элемент управления картами.
- MapActivity. Базовый класс, при наследовании которого создается новая Активность, содержащая MapView. Класс MapActivity управляет жизненным циклом приложения и фоновыми задачами, необходимыми для отображения карт. Поэтому элементы MapView используются только внутри Активности, наследованной от MapActivity.
- Overlay. Этот класс требуется для добавления аннотаций к вашей карте. Используя Наложения, можно рисовать произвольное количество слоев на элементе Canvas, и они будут отображены поверх MapView.
- MapController. Используется для управления картой, в том числе указания местоположения и уровня масштаба.
- MyLocationOverlay. Специальное Наложение, которое может быть применено для показа текущего местоположения и ориентации устройства.
- ItemizedOverlays и OverlayItems. Работают совместно, позволяют создавать слой меток, которые будут отображаться с помощью объектов Drawable и соответствующего текста.

Получение собственного ключа к API для карт

Чтобы использовать MapView в вашем приложении, сперва нужно получить ключ для API на сайте для разработчиков под Android http://code.google.com/android/maps-api-signup.html.

Без ключа для API MapView не загрузит необходимые для отображения карты фрагменты.

Для получения ключа нужно указать MD5-слепок сертификата, использующегося для подписи приложения. Как правило, подписывать свои приложения вы будете с помощью двух сертификатов — отладочного, созданного по умолчанию, и реального промышленного. В следующих разделах рассказывается, как получить слепок MD5 для каждого подписанного сертификата, используемого в приложении.

Получение отладочного слепка MD5

Если вы применяете Eclipse с дополнением ADT для отладки приложения, он уже поставляется с отладочным сертификатом по умолчанию. Для того чтобы отображать карту во время отладки, нужно получить ключ к API карт, зарегистрированный через слепок MD5 отладочного сертификата.

Вы можете найти путь к хранилищу ключей в поле ввода `Default Debug Keystore`, открыв меню Windows ▸ Preferences ▸ Android ▸ build. Как правило, хранилище ключей для отладки находится в следующих местах на диске в зависимости от платформы:

- Windows Vista: `\users\<username>\.android\debug.keystore`;
- Windows XP: `\Documents and Settings\<username>\.android\debug.keystore`;
- Linux или Mac: `~/.android/debug.keystore`.

ПРИМЕЧАНИЕ

Каждый компьютер, который вы используете для разработки, получит собственные сертификат и слепок MD5. Если вы хотите отлаживать и разрабатывать приложение, которое работает с картами, используя сразу несколько компьютеров, нужно сгенерировать и задействовать несколько ключей к API.

Чтобы найти слепок MD5 вашего отладочного сертификата, воспользуйтесь командой keytool, входящей в установочный набор Java:

```
keytool -list -alias androiddebugkey -keystore <keystore_location>.
keystore -storepass android -keypass android
```

Получение промышленного слепка MD5

Прежде чем скомпилировать и подписать приложение для выпуска, нужно получить ключ к API карт, используя слепок MD5 для вашего промышленного сертификата.

Найдите слепок MD5 с помощью команды keytool, укажите параметр -list, имя хранилища с ключами и псевдоним, который использовали при подписи своего приложения.

```
keytool -list -alias my-android-alias -keystore my-android-keystore
```

Чтобы получить слепок MD5, нужно ввести пароли от хранилища ключей и псевдонима.

Создание Активности, использующей картографию

Если хотите использовать карты в своем приложении, то должны наследовать класс MapActivity. Разметка нового класса обязана включать MapView для отображения карт Google Maps. Картографическая библиотека в Android — нестандартный пакет. Как дополнительный API, она должна быть явно внесена в манифест приложения перед использованием. Добавьте библиотеку в ваш манифест с помощью тега `uses-library`, включив его внутрь узла application, как показано в следующем фрагменте кода XML:

```
<uses-library android:name="com.google.android.maps"/>
```

ПРИМЕЧАНИЕ

Пакет с картами не входит в стандартный открытый проект Android. Он предоставляется в рамках SDK Android компанией Google и доступен на большинстве устройств под управлением Android. Однако следует помнить, что устройство может и не содержать данного пакета, поскольку он, как отмечалось, нестандартный.

MapView загружает фрагменты карты при необходимости. В связи с этим он косвенно требует полномочий для использования Интернета. Поэтому нужно добавить тег `<uses-permission>`, содержащий полномочие INTERNET, в манифест вашего приложения, как показано ниже:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Добавив библиотеку и настроив полномочия, вы готовы к созданию новой Активности, использующей возможности картографии.

Элемент управления MapView может функционировать только в сочетании с Активностью — потомком MapActivity. Переопределите метод onCreate, чтобы вывести на экран MapView, и isRouteDisplayed, который должен

возвращать true, если Активность будет выводить информацию о маршрутах (например, направление движения).

В листинге 8.6 показан каркас для создания потомка MapActivity.

Листинг 8.6. Каркас для MapActivity

```
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import android.os.Bundle;

public class MyMapActivity extends MapActivity {
  private MapView mapView;

  private MapController mapController;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.map_layout);
    mapView = (MapView)findViewById(R.id.map_view);
  }

  @Override
  protected boolean isRouteDisplayed() {
    // ВАЖНО: Этот метод должен возвращать true, если ваша Активность
    // показывает направления движения. В ином случае он должен
    // вернуть false.
    return false;
  }
}
```

Соответствующий файл с разметкой, в который добавляется MapView, показан в листинге 8.7. Обратите внимание, что вы должны включить в свое приложение ключ к API карт (как было показано ранее в этой главе), чтобы иметь возможность использовать MapView.

Листинг 8.7. Ресурс с разметкой MapActivity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <com.google.android.maps.MapView
    android:id="@+id/map_view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="mymapapikey"
  />
</LinearLayout>
```

На рис. 8.4 представлен пример простой Активности с картой.



Рис. 8.4.

ПРИМЕЧАНИЕ

В настоящее время для одного приложения в Android поддерживается только одна Активность MapActivity и только один экземпляр MapView.

Настройка и использование элементов MapView

Класс MapView отображает карту Google Maps: поддерживает несколько параметров, которые учитываются при отображении карты.

По умолчанию объект MapView выводит стандартную схематическую карту, как показано на рис. 8.4. В дополнение вы можете выбрать спутниковый и транспортный режимы отображения, как в следующем фрагменте:

```
mapView.setSatellite(true);
mapView.setStreetView(true);
mapView.setTraffic(true);
```

Можно также делать запросы к MapView, чтобы получить текущий и максимальный доступный уровни масштаба наряду с центральной точкой

на карте, а также широтой и долготой отображаемого участка карты (в десятичных градусах). Последнее (демонстрируется в следующем фрагменте кода) пригодится для поисков с помощью геокодировщика на ограниченных участках.

```
int maxZoom = mapView.getMaxZoomLevel();
GeoPoint center = mapView.getMapCenter();
int latSpan = mapView.getLatitudeSpan();
int longSpan = mapView.getLongitudeSpan();
```

При желании отобразите стандартные элементы управления масштабом, воспользовавшись методом setBuiltInZoomControls.

```
mapView.setBuiltInZoomControls(true);
```

Использование объекта MapController

Задействуйте MapController для наведения и масштабирования MapView. Вы можете получить ссылку на элементы управления MapView с помощью метода getController.

```
MapController mapController = myMapView.getController();
```

Местоположения на карте в картографических классах Android представлены объектами GeoPoint, которые содержат широту и долготу, измеряемые в микроградусах. Чтобы перевести градусы в микроградусы, умножьте значение на миллион (1 000 000).

Прежде чем начинать использовать значения широты и долготы, хранящиеся в объекте Location, полученном из геолокационных сервисов, необходимо перевести их в микроградусы и передать объекту GeoPoint.

```
Double lat = 37.422006*1E6;
Double lng = -122.084095*1E6;
GeoPoint point = new GeoPoint(lat.intValue(), lng.intValue());
```

Отцентрируйте и масштабируйте MapView с помощью методов setCenter и setZoom, доступных из объекта MapController, принадлежащего MapView.

```
mapController.setCenter(point);
mapController.setZoom(1);
```

При использовании setZoom значение 1 соответствует самому малому (удаленному) масштабу, а 21 — самому большому (приближенному).

Фактический масштаб, доступный для конкретной местности, зависит от разрешения карт Google Maps и качества снимков этого района. Вы также можете использовать методы zoomIn и zoomOut, чтобы изменять масштаб на одно значение в ту или иную строну.

Метод setCenter установит новое текущее местоположение, для показа плавного перехода воспользуйтесь методом animateTo.

```
mapController.animateTo(point);
```

Добавление карты в приложение Where Am I?

В следующем примере[1] проект Where Am I? опять будет дополнен. На этот раз вы добавите в него картографические возможности, превратив Активность приложения в MapActivity. По мере изменения местоположения устройства карта станет автоматически центрироваться на новой позиции.

1. Начните с добавления тега <uses-permission> в манифест приложения для получения доступа в Интернет. Импортируйте библиотеку с картами Android в теге <application>.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.paad.whereami">
  <application
    android:icon="@drawable/icon">
    <uses-library android:name="com.google.android.maps"/>
    <activity
      android:name=".WhereAmI"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
</manifest>
```

2. Измените класс-родитель для WhereAmI на MapActivity вместо Activity. Переопределите метод isRouteDisplayed. Поскольку эта Активность не будет показывать маршрут, можете вернуть значение false.

```
public class WhereAmI extends MapActivity {
  @Override
  protected boolean isRouteDisplayed() {
    return false;
  }
  [ . . . ранее добавленный код Активности . . . ]
}
```

3. Отредактируйте разметку в ресурсе main.xml, включив в нее элемент MapView и используя при этом полное название класса. Вам нужно

[1] Все фрагменты кода в этом примере — часть проекта Where Am I? из главы 8, их можно загрузить с сайта Wrox.com.

получить ключ к API карт, чтобы добавить атрибут `android:apikey` в узел com.google.android.maps.MapView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:id="@+id/myLocationText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
  />
  <com.google.android.maps.MapView
    android:id="@+id/myMapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="myMapKey"
  />
</LinearLayout>
```

4. Теперь при запуске приложения должны отобразиться оригинальный текст из геолокационного сервиса и MapView под ним, как показано на рис. 8.5.

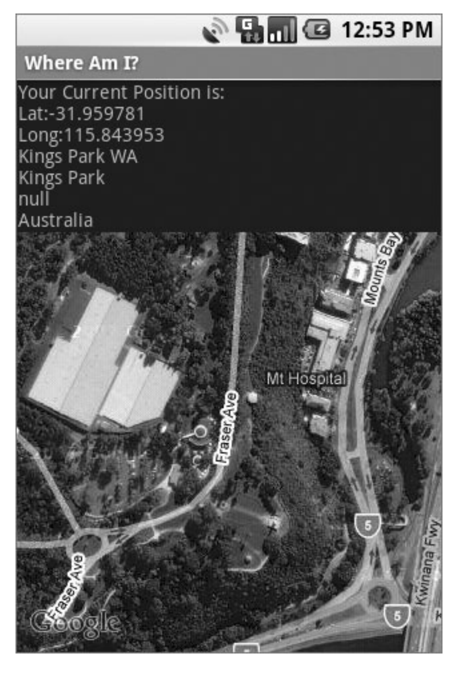


Рис. 8.5.

5. Настройте MapView и сохраните ссылку на его объект MapController в виде переменной. Установите параметры отображения MapView, чтобы показать спутниковые и схематичные режимы, а также увеличьте масштаб, чтобы приблизить карту.

```
MapController mapController;

@Override
public void onCreate(Bundle savedInstance State) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  // Получите ссылку на MapView
  MapView myMapView = (MapView)findViewById(R.id.myMapView);
  // Получите объект MapController, принадлежащий MapView
  mapController = myMapView.getController();

  // Настройте параметры отображения карты
  myMapView.setSatellite(true);
  myMapView.setStreetView(true);
  myMapView.displayZoomControls(false);

  // Увеличьте масштаб
  mapController.setZoom(17);

  LocationManager locationManager;
  String context = Context.LOCATION_SERVICE;
  locationManager = (LocationManager)getSystemService(context);

  Criteria criteria = new Criteria();

  criteria.setAccuracy(Criteria.ACCURACY_FINE);
  criteria.setAltitudeRequired(false);
  criteria.setBearingRequired(false);
  criteria.setCostAllowed(true);
  criteria.setPowerRequirement(Criteria.POWER_LOW);
  String provider = locationManager.getBestProvider(criteria, true);

  Location location =
    locationManager.getLastKnownLocation(provider);

  updateWithNewLocation(location);

  locationManager.requestLocationUpdates(provider, 2000, 10,
                                         locationListener);
}
```

6. В завершение отредактируйте метод updateWithNewLocation, чтобы отцентрировать карту согласно текущему местоположению с помощью MapController.

```
private void updateWithNewLocation(Location location) {
  String latLongString;
  TextView myLocationText;
  myLocationText = (TextView)findViewById(R.id.myLocationText);
```

```
    String addressString = "No address found";

    if (location != null) {
      // Обновите местоположение на карте.
      Double geoLat = location.getLatitude()*1E6;
      Double geoLng = location.getLongitude()*1E6;
      GeoPoint point = new GeoPoint(geoLat.intValue(),
      geoLng.intValue());

      mapController.animateTo(point);

      double lat = location.getLatitude();
      double lng = location.getLongitude();
      latLongString = "Lat:" + lat + "\nLong:" + lng;

      double latitude = location.getLatitude();
      double longitude = location.getLongitude();

      Geocoder gc = new Geocoder(this, Locale.getDefault());
      try {
        List<Address> addresses = gc.getFromLocation(latitude, longitude,
  1);
        StringBuilder sb = new StringBuilder();
        if (addresses.size() > 0) {
        Address address = addresses.get(0);

        for (int i = 0; i < address.getMaxAddressLineIndex(); i++)
          sb.append(address.getAddressLine(i)).append("\n");

          sb.append(address.getLocality()).append("\n");
          sb.append(address.getPostalCode()).append("\n");
          sb.append(address.getCountryName());
        }

        addressString = sb.toString();
      } catch (IOException e) {}
    } else {
    latLongString = "No location found";
    }
    myLocationText.setText("Your Current Position is:\n" +
                           latLongString + "\n" + addressString);
  }
```

Создание и использование Наложений

Наложения позволяют добавлять к MapView аннотации и обработку нажатий. На каждом Наложении с помощью Холста можно рисовать двумерные примитивы, включая текст, линии, изображения и геометрические фигуры; после этого содержимое Холста накладывается на MapView.

Вы можете добавить несколько Наложений на одну карту. Все они размещаются в виде слоев, поэтому новые могут перекрывать старые. Пользовательские нажатия передаются через стек и обрабатываются Наложением или зарегистрированными обработчиками событий в самом элементе MapView.

Создание новых Наложений

Каждое Наложение — Холст с прозрачным фоном, который наслаивается на MapView и используется для обработки событий при нажатии.

Для добавления Наложения создайте новый класс, наследующий Overlay. Переопределите метод draw, чтобы нарисовать аннотации, которые хотите внести. Переопределите обработчик onTap, чтобы реагировать на нажатия (происходит, когда пользователь нажимает на аннотации, добавленные данным Наложением).

В листинге 8.8 показан каркас для создания нового Наложения, которое может рисовать аннотации и обрабатывать пользовательские нажатия.

Листинг 8.8. Создание нового Наложения

```
import android.graphics.Canvas;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

public class MyOverlay extends Overlay {
  @Override
  public void draw(Canvas canvas, MapView mapView, boolean shadow) {
    if (shadow == false) {
      [ . . . Рисуем аннотации на главном слое карты . . . ]
    }
    else {
      [ . . . Рисуем аннотации на затененном слое . . . ]
    }
  }

  @Override
  public boolean onTap(GeoPoint point, MapView mapView) {
    // Верните true, если нажатие на экран обрабатывается данным Наложением
    return false;
  }
}
```

Знакомство с проекциями

Холст для рисования аннотаций на Наложении — это обычный объект Canvas, видимая поверхность экрана. Чтобы добавить аннотации, основанные на физическом местоположении, нужно преобразовать географические точки в экранные координаты.

Класс Projection позволяет переводить значения широты/долготы (хранятся в объекте GeoPoint) в координаты пиксела на экране (хранятся в виде объекта Point).

Проекция карты может меняться при перерисовке, поэтому желательно каждый раз получать новый экземпляр проекции. Вызовите метод getProjection, чтобы получить проекцию для MapView.

```
Projection projection = mapView.getProjection();
```

Используйте методы fromPixel и toPixel, чтобы переводить GeoPoint в Point и наоборот.

Для повышения производительности лучше передавать в метод toPixel объект Point с последующим его заполнением (а не получать возвращаемое значение), как показано в листинге 8.9.

Листинг 8.9. Использование проекций

```
Point myPoint = new Point();
// В координаты экрана
projection.toPixels(geoPoint, myPoint);
// В географические координаты GeoPoint
projection.fromPixels(myPoint.x, myPoint.y);
```

Рисование на Наложении с помощью Холста

Чтобы рисовать на Наложении с помощью Холста, нужно переопределить обработчик draw, принадлежащий объекту Overlay.

Объект Canvas, передающийся в этот метод, выступает поверхностью, на которой вы будете рисовать свои аннотации. Применяется та же методика, что и при создании нестандартных пользовательских интерфейсов для Представлений, с которыми вы познакомились в главе 4. Объект Canvas включает методы для рисования двумерных примитивов на вашей карте (в том числе линии, текст, различные фигуры, эллипсы, изображения и т. д.). Используйте объект Paint, чтобы определить стили и цвета.

В листинге 8.10 используется проекция для отрисовки текста и эллипса поверх заданной местности.

Листинг 8.10. Простое Наложение на карте

```
@Override
public void draw(Canvas canvas, MapView mapView, boolean shadow) {
  Projection projection = mapView.getProjection();

  Double lat = -31.960906*1E6;
  Double lng = 115.844822*1E6;
  GeoPoint geoPoint = new GeoPoint(lat.intValue(), lng.intValue());

  if (shadow == false) {
    Point myPoint = new Point();
    projection.toPixels(geoPoint, myPoint);

    // Создайте и настройте вашу кисть для рисования
    Paint paint = new Paint();
    paint.setARGB(250, 255, 0, 0);
    paint.setAntiAlias(true);
    paint.setFakeBoldText(true);

    // Создайте окружность
    int rad = 5;
```

Продолжение ⇗

Листинг 8.10 (продолжение)

```
      RectF oval = new RectF(myPoint.x-rad, myPoint.y-rad,
                             myPoint.x+rad, myPoint.y+rad);

      // Нарисуйте на Холсте текст и эллипс
      canvas.drawOval(oval, paint);
      canvas.drawText("Red Circle", myPoint.x+rad, myPoint.y, paint);
    }
  }
```

ПРИМЕЧАНИЕ

Если вам нужны более продвинутые функции для рисования, смотрите главу 11 — в ней описаны градиенты, начертания и фильтры.

Обработка нажатий на карте

Чтобы обрабатывать нажатия на карте (пользовательские нажатия), переопределите обработчик onTap внутри наследованного от Overlay класса.

Обработчик onTap принимает два параметра:

- объект GeoPoint, который содержит широту/долготу нажатой точки;
- объект MapView, после нажатия на который сгенерировалось это событие.

При переопределении метода onTap учитывайте, что он должен возвращать true, если сам обрабатывает конкретное нажатие, и false, если дает возможность Наложению проводить обработку данного события. Все это показано в листинге 8.11:

Листинг 8.11. Обработка нажатий на карте

```
@Override
public boolean onTap(GeoPoint point, MapView mapView) {
  // Perform hit test to see if this overlay is handling the click
  if ([ . . . выполнить проверку нажатия  . . . ]) {
    [ . . . выполнить обработку нажатия . . . ]
  return true;
  }

  // Если обрабатывать не нужно, верните false
  return false;
}
```

Добавление и удаление Наложений

Каждый объект MapView содержит список Наложений, которые отображаются в данный момент. Вы можете получить ссылку на этот список, вызвав метод getOverlays, как показано в следующем фрагменте:

```
List<Overlay> overlays = mapView.getOverlays();
```

Элементы этого списка добавляются и удаляются синхронно и в безопасном потоковом режиме, поэтому вы можете спокойно изменять перечень и делать к нему запросы. Вы также должны перебирать элементы списка внутри блока синхронизации, который в свою очередь синхронизируется с объектом List.

Чтобы добавить Наложение на MapView, создайте новый экземпляр класса Overlay и внесите его в список, как показано в следующем фрагменте:

```
List<Overlay> overlays = mapView.getOverlays();
MyOverlay myOverlay = new MyOverlay();
overlays.add(myOverlay);
mapView.postInvalidate();
```

Добавленное Наложение будет выведено при следующей перерисовке MapView, поэтому не помешает вызвать метод postInvalidate после изменения списка, чтобы обновить изменения на карте.

Добавление аннотаций в приложение Where Am I?

Эти заключительные изменения в проекте Where Am I? создают и добавляют новое Наложение, которое выводит белую окружность над текущим местоположением устройства[1].

1. Начните с создания нового класса MyPositionOverlay в проекте Where Am I?.

```
package com.paad.whereami;

import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.RectF;
import android.location.Location;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.Projection;

public class MyPositionOverlay extends Overlay {

  @Override
  public void draw(Canvas canvas, MapView mapView, boolean shadow) {
  }

  @Override
  public boolean onTap(GeoPoint point, MapView mapView) {
    return false;
  }
}
```

[1] Все фрагменты кода в этом примере — часть проекта Where Am I? из главы 8, их можно загрузить с сайта Wrox.com.

2. Создайте новое поле для хранения текущего местоположения, добавьте геттер и сеттер.

```
Location location;

public Location getLocation() {
  return location;
}
public void setLocation(Location location) {
  this.location = location;
}
```

3. Переопределите метод draw, чтобы нарисовать небольшую белую окружность над текущим местоположением.

```
private final int mRadius = 5;

@Override
public void draw(Canvas canvas, MapView mapView, boolean shadow) {
  Projection projection = mapView.getProjection();

  if (shadow == false) {
    // Получите текущее местоположение
    Double latitude = location.getLatitude()*1E6;
    Double longitude = location.getLongitude()*1E6;
    GeoPoint geoPoint;
    geoPoint = new
      GeoPoint(latitude.intValue(),longitude.intValue());

    // Преобразуйте местоположение в экранные координаты
    Point point = new Point();
    projection.toPixels(geoPoint, point);

    RectF oval = new RectF(point.x - mRadius, point.y - mRadius,
                           point.x + mRadius, point.y + mRadius);

    // Подготовьте объект Paint
    Paint paint = new Paint();
    paint.setARGB(250, 255, 255, 255);
    paint.setAntiAlias(true);
    paint.setFakeBoldText(true);

    Paint backPaint = new Paint();
    backPaint.setARGB(175, 50, 50, 50);
    backPaint.setAntiAlias(true);

    RectF backRect = new RectF(point.x + 2 + mRadius,
                               point.y - 3*mRadius,
                               point.x + 65, point.y + mRadius);

    // Нарисуйте отметку
    canvas.drawOval(oval, paint);
    canvas.drawRoundRect(backRect, 5, 5, backPaint);
    canvas.drawText("Here I Am",
```

```
                  point.x + 2*mRadius, point.y,
                  paint);
  }
  super.draw(canvas, mapView, shadow);
}
```

4. Перейдите к классу **Активности** WhereAmI и добавьте MyPositionOverlay в объект MapView.

   Начните с добавления нового поля для хранения MyPositionOverlay, затем переопределите метод onCreate, чтобы создать новый экземпляр класса, и внесите его в список **Наложений** объекта MapView.

```
MyPositionOverlay positionOverlay;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  MapView myMapView = (MapView)findViewById(R.id.myMapView);
  mapController = myMapView.getController();

  myMapView.setSatellite(true);
  myMapView.setStreetView(true);
  myMapView.displayZoomControls(false);

  mapController.setZoom(17);

  // Добавьте объект MyPositionOverlay
  positionOverlay = new MyPositionOverlay();
  List<Overlay> overlays = myMapView.getOverlays();
  overlays.add(positionOverlay);

  LocationManager locationManager;
  String context = Context.LOCATION_SERVICE;
  locationManager = (LocationManager)getSystemService(context);

  Criteria criteria = new Criteria();
  criteria.setAccuracy(Criteria.ACCURACY_FINE);
  criteria.setAltitudeRequired(false);
  criteria.setBearingRequired(false);
  criteria.setCostAllowed(true);
  criteria.setPowerRequirement(Criteria.POWER_LOW);
  String provider = locationManager.getBestProvider(criteria, true);

  Location location = locationManager.getLastKnownLocation(provider);

  updateWithNewLocation(location);

  locationManager.requestLocationUpdates(provider, 2000, 10,
                                         locationListener);
}
```

5. В завершение отредактируйте метод updateWithNewLocation, передавая новое местоположение в объект MyPositionOverlay.

```
private void updateWithNewLocation(Location location) {
  String latLongString;
  TextView myLocationText;

  myLocationText = (TextView)findViewById(R.id.myLocationText);
  String addressString = "No address found";

  if (location != null) {
    // Обновите отметку для текущего местоположения
    positionOverlay.setLocation(location);

    // Обновите местоположение на карте.
    Double geoLat = location.getLatitude()*1E6;
    Double geoLng = location.getLongitude()*1E6;
    GeoPoint point = new GeoPoint(geoLat.intValue(),
                                  geoLng.intValue());

    mapController.animateTo(point);

    double lat = location.getLatitude();
    double lng = location.getLongitude();
    latLongString = "Lat:" + lat + "\nLong:" + lng;

    double latitude = location.getLatitude();
    double longitude = location.getLongitude();

    Geocoder gc = new Geocoder(this, Locale.getDefault());
    try {
      List<Address> addresses = gc.getFromLocation(latitude,
                                             longitude, 1);
      StringBuilder sb = new StringBuilder();
      if (addresses.size() > 0) {
      Address address = addresses.get(0);

      for (int i = 0; i < address.getMaxAddressLineIndex(); i++)
        sb.append(address.getAddressLine(i)).append("\n");

        sb.append(address.getLocality()).append("\n");
        sb.append(address.getPostalCode()).append("\n");
        sb.append(address.getCountryName());
      }
      addressString = sb.toString();
    } catch (IOException e) {}
  } else {
    latLongString = "No location found";
  }
  myLocationText.setText("Your Current Position is:\n" +
                         latLongString + "\n" + addressString);
}
```

При запуске приложение отобразит текущее местоположение вашего устройства с помощью белой окружности и вспомогательного текста, как показано на рис. 8.6.



Рис. 8.6.

ПРИМЕЧАНИЕ

Приведенный метод далеко не самый предпочтительный при выводе вашей текущей позиции на карте. Данная функциональность уже реализована в Android с помощью класса MyLocationOverlay. Если хотите отображать и обновлять ваше текущее местоположение, вместо реализации, описанной выше, рассмотрите использование (или наследование) этого класса, как показано в следующем разделе.

Знакомство с MyLocationOverlay

Класс MyLocationOverlay — специальный вид Наложения, спроектированный для того, чтобы показывать ваше текущее местоположение и ориентацию на объекте MapView.

Чтобы применить MyLocationOverlay, требуется создать новый экземпляр, передав ему в качестве параметров объект Context приложения и элемент MapView, и добавить его в список Наложений в MapView, как показано далее:

```
List<Overlay> overlays =
  mapView.getOverlays();
MyLocationOverlay myLocationOverlay =
  new MyLocationOverlay(this, mapView);
overlays.add(myLocationOverlay);
```

Вы можете использовать MyLocationOverlay для отображения текущих местоположения (в виде синей мигающей отметки) и ориентации (в виде компаса на карте).

В следующем фрагменте показано, каким образом можно включить отображение и компаса, и отметки. В данном примере в виде параметра передается MapController, принадлежащий объекту MapView, что позволяет Наложению автоматически прокручивать карту, если отметка выходит за пределы экрана.

```
myLocationOverlay.enableCompass();
myLocationOverlay.enableMyLocation(mapView.getMapController());
```

Знакомство с Детализированными наложениями и с объектом OverlayItem

Объекты OverlayItem нужны, чтобы обеспечить простую функциональность вашему объекту MapView с помощью класса ItemizedOverlay.

Класс ItemizedOverlay предоставляет удобные и быстрые вызовы для добавления отметок на карту, позволяет задавать для них изображения и привязывать текст к конкретной географической позиции. Экземпляр ItemizedOverlay контролирует отрисовку, размещение, обработку нажатий, фокусировку и оптимизацию слоев для каждой отметки класса OverlayItem.

Чтобы добавить слой с отметками ItemizedOverlay на вашу карту, начните с создания нового класса, наследующего ItemizedOverlay<OverlayItem>, как показано в листинге 8.12.

ПРИМЕЧАНИЕ

ItemizedOverlay — обобщенный класс, помогающий создавать потомков, основанных на любом производном от OverlayItem классе.

Для начала необходимо вызвать конструктор родительского класса, определив границы для вашей отметки по умолчанию. Затем нужно обратиться к методу populate для инициирования создания каждого элемента OverlayItem. Этот метод должен вызываться всякий раз, когда используемые данные меняются.

Внутри реализации класса MyItemizedOverlay переопределите метод size. Он должен возвращать количество отметок для вывода на экран. Переопределите также метод createItem, чтобы создавать новые элементы, основываясь на индексе каждой отметки.

Листинг 8.12. Создание нового Детализированного наложения

```
import android.graphics.drawable.Drawable;
import com.google.android.maps.GeoPoint;
```

```
import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.OverlayItem;

public class MyItemizedOverlay extends ItemizedOverlay<OverlayItem> {

  public MyItemizedOverlay(Drawable defaultMarker) {
    super(boundCenterBottom(defaultMarker));
    populate();
  }

  @Override
  protected OverlayItem createItem(int index) {
    switch (index) {
      case 1:
        Double lat = 37.422006*1E6;
        Double lng = -122.084095*1E6;
        GeoPoint point = new GeoPoint(lat.intValue(), lng.intValue());

        OverlayItem oi;
        oi = new OverlayItem(point, "Marker", "Marker Text");
        return oi;
    }
    return null;
  }

  @Override
  public int size() {
    // Верните количество отметок в коллекции
    return 1;
  }
}
```

Чтобы добавить реализацию ItemizedOverlay на вашу карту, выполните новый экземпляр этого класса (передав ему Drawable для использования в каждой отметке) и добавьте его в список Наложений, принадлежащий MapView.

```
List<Overlay> overlays = mapView.getOverlays();
MyItemizedOverlay markers = new
  MyItemizedOverlay(r.getDrawable(R.drawable.marker));
overlays.add(markers);
```

ВНИМАНИЕ

Метки на карте, размещенные с помощью Детализированного наложения, используют механизм состояний для уведомления о том, выбраны ли они. Задействуйте объект StateListDrawable, описанный в главе 4, чтобы сигнализировать о выборе метки.

В листинге 8.12 список элементов Наложения статичен и прописан в коде. На деле элементы вашего Наложения будут храниться в динамическом списке ArrayList, куда вы сможете добавлять элементы, а также удалять их во время выполнения программы.

В листинге 8.13 показан каркас класса для динамической реализации ItemizedOverlay, использующий ArrayList и поддерживающий добавление и удаление элементов во время выполнения программы.

Листинг 8.13. Каркас для динамического Детализированного наложения

```
public class MyDynamicItemizedOverlay extends
ItemizedOverlay<OverlayItem> {

  private ArrayList<OverlayItem> items;

  public MyDynamicItemizedOverlay(Drawable defaultMarker) {
    super(boundCenterBottom(defaultMarker));
    items = new ArrayList<OverlayItem>();
    populate();
  }

  public void addNewItem(GeoPoint location, String markerText,
                         String snippet) {
    items.add(new OverlayItem(location, markerText, snippet));
    populate();
  }

  public void removeItem(int index) {
    items.remove(index);
    populate();
  }

  @Override
  protected OverlayItem createItem(int index) {
    return items.get(index);
  }

  @Override
  public int size() {
    return items.size();
  }
}
```

Добавление Представлений на карту

Вы можете добавить на MapView любой объект, наследованный от View (включая разметку и другие Группы представлений), прикрепляя их к определенной позиции на экране или к географическому местоположению.

В последнем случае Представление последует за позицией на карте, к которой его прикрепили, демонстрируя поведение, аналогичное интерактивной отметке. Как более ресурсоемкое решение, оно обычно используется для показа всплывающих сообщений с информацией (часто применяется в гибридных проектах для предоставления детальных данных при нажатии на маркер).

Вы реализуете оба механизма для добавления Представлений на карту, вызывая метод addView, принадлежащий MapView. Как правило, это делается внутри методов onCreate или onRestore при реализации Активности MapActivity. Передайте Представление, которое хотите добавить на карту, и параметры разметки, которые желаете использовать.

Параметры MapView.LayoutParams, которые передаются в метод addView, определяют место на карте, куда Представление попадет.

Чтобы добавить новое Представление на карту относительно экрана, задайте новый объект MapView.LayoutParams, включив в него аргументы, отвечающие за его высоту и ширину, экранные координаты x/y, а также способ выравнивания для позиционирования, как показано в листинге 8.14.

Листинг 8.14. Добавление Представления на карту

```
int y = 10; int x = 10;

EditText editText1 = new EditText(getApplicationContext());
editText1.setText("Screen Pinned");

MapView.LayoutParams screenLP;
screenLP = new MapView.LayoutParams(MapView.LayoutParams.WRAP_CONTENT,
                                    MapView.LayoutParams.WRAP_CONTENT,
                                    x, y,
                                    MapView.LayoutParams.TOP_LEFT);
mapView.addView(editText1, screenLP);
```

Чтобы закрепить Представление на карте относительно физического местоположения, передайте конструктору LayoutParams четыре аргумента: высоту, ширину, объект GeoPoint, к которому будет привязано Представление, а также выравнивание разметки, как показано в листинге 8.15.

Листинг 8.15. Закрепление Представления за географическим местоположением

```
Double lat = 37.422134*1E6;
Double lng = -122.084069*1E6;
GeoPoint geoPoint = new GeoPoint(lat.intValue(), lng.intValue());

MapView.LayoutParams geoLP;
geoLP = new MapView.LayoutParams(MapView.LayoutParams.WRAP_CONTENT,
                                 MapView.LayoutParams.WRAP_CONTENT,
                                 geoPoint,
                                 MapView.LayoutParams.TOP_LEFT);

EditText editText2 = new EditText(getApplicationContext());
editText2.setText("Location Pinned");

mapView.addView(editText2, geoLP);
```

После этого первый элемент TextView останется неподвижным на карте в левом верхнем углу, а второй будет привязан к определенной географической позиции.

Чтобы удалить Представление из MapView, вызовите метод removeView, передав ему в качестве аргумента экземпляр Представления, которое хотите удалить, как показано ниже:

```
mapView.removeView(editText2);
```

Добавление картографических возможностей в проект Earthquake

Представленное пошаговое руководство продемонстрирует, как для проекта Earthquake, который вы начали в главе 5, создать Активность, использующую картографические возможности. Новая Активность MapActivity[1] будет отображать карту с последними землетрясениями, используя методику, с которой вы познакомились в этой главе.

1. Создайте новый ресурс с разметкой `earthquake_map.xml`, который содержит MapView. Убедитесь, что вы не забыли добавить атрибуты `android:id` и `android:apikey` (последний содержит ключ к API карт для Android).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <com.google.android.maps.MapView
    android:id="@+id/map_view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="myapikey"
  />
</LinearLayout>
```

2. Создайте новую Активность EarthquakeMap, которая будет наследовать MapActivity. Используйте метод setContentView внутри onCreate, чтобы получить доступ к ресурсу earthquake_map, созданному на предыдущем шаге.

```
package com.paad.earthquake;

import android.os.Bundle;
import com.google.android.maps.MapActivity;

public class EarthquakeMap extends MapActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
```

[1] Все фрагменты кода в этом примере — часть проекта Earthquake из главы 8, их можно загрузить с сайта Wrox.com.

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.earthquake_map);
  }

  @Override
  protected boolean isRouteDisplayed() {
    return false;
  }
}
```

3. Отредактируйте манифест приложения, включив в него новую Активность EarthquakeMap и импортировав библиотеку для работы с картами.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.paad.earthquake">
  <application android:icon="@drawable/icon">
    <activity
      android:name=".Earthquake"
      android:label="@string/app_name">
      <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".Preferences"
              android:label="Earthquake Preferences"/>
    <activity android:name=".EarthquakeMap"
              android:label="View Earthquakes"/>
    <provider android:name=".EarthquakeProvider"
              android:authorities="com.paad.provider.earthquake" />
    <uses-library android:name="com.google.android.maps"/>
  </application>
  <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

4. Добавьте новый элемент в меню Активности Earthquake, при активизации которого будет отображаться EarthquakeMap.

   4.1. Начните с добавления новой строки для меню в ресурс `strings.xml`.

```
<?xml version="1.0" encoding="autf-8"?>
<resources>
  <string name="app_name">Earthquake</string>
  <string name="quake_feed">
    http://earthquake.usgs.gov/eqcenter/catalogs/1day-M2.5.xml
  </string>
  <string name="menu_update">Refresh Earthquakes</string> <string
name="auto_update_prompt">Auto Update?</string>
  <string name="update_freq_prompt">Update Frequency</string>
  <string name="min_quake_mag_prompt">
    Minimum Quake Magnitude
  </string>
  <string name="menu_preferences">Preferences</string>
  <string name="menu_earthquake_map">Earthquake Map</string>
</resources>
```

4.2. Добавьте новый идентификатор меню, отредактируйте обработчик onCreateOptionsMenu, чтобы внести новый элемент. Он должен использовать текст, который вы определили в пункте 4.1, и при выборе активизировать для явного вызова Активности EarthquakeMap.

```
static final private int MENU_EARTHQUAKE_MAP = Menu.FIRST+2;

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  menu.add(0, MENU_UPDATE, Menu.NONE, R.string.menu_update);
  menu.add(0, MENU_PREFERENCES, Menu.NONE,
           R.string.menu_preferences);
  Intent startMap = new Intent(this, EarthquakeMap.class);
  menu.add(0, MENU_EARTHQUAKE_MAP,
           Menu.NONE,
           R.string.menu_earthquake_map).setIntent(startMap);
  return true;
}
```

5. Теперь создайте новый класс EarthquakeOverlay, наследующий Overlay. Он будет выводить позицию и магнитуду каждого землетрясения на карте.

```
package com.paad.earthquake;

import java.util.ArrayList;
import android.database.Cursor;
import android.database.DataSetObserver;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.RectF;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.Projection;

public class EarthquakeOverlay extends Overlay {
  @Override
  public void draw(Canvas canvas, MapView mapView, boolean shadow) {
    Projection projection = mapView.getProjection();

    if (shadow == false) {
      // TODO: Отобразить землетрясения
    }
  }
}
```

5.1. Добавьте новый конструктор, который принимает объект Cursor, ссылающийся на данные о текущем землетрясении, и сохраните его в виде поля класса.

```
Cursor earthquakes;

public EarthquakeOverlay(Cursor cursor, ContentResolver resolver) {
  super();

  earthquakes = cursor;
}
```

5.2. Создайте новый метод refreshQuakeLocations, который перебирает данные из результирующего Курсора и извлекает местоположение каждого землетрясения в виде широты и долготы, прежде чем сохранить координаты в список объектов GeoPoint.

```
ArrayList<GeoPoint> quakeLocations;

private void refreshQuakeLocations() {
  if (earthquakes.moveToFirst())
    do {
      Double lat =
        earthquakes.getFloat(EarthquakeProvider.LATITUDE_COLUMN) * 1E6;
      Double lng =
        earthquakes.getFloat(EarthquakeProvider.LONGITUDE_COLUMN) * 1E6;

      GeoPoint geoPoint = new GeoPoint(lng.intValue(),
                                       lat.intValue());
      quakeLocations.add(geoPoint);

    } while(earthquakes.moveToNext());
}
```

5.3. Вызовите метод refreshQuakeLocations из конструктора Наложения. Также зарегистрируйте объект DataSetObserver из результирующего Курсора, чтобы обновлять список местоположения землетрясений при изменениях в Cursor.

```
public EarthquakeOverlay(Cursor cursor) {
  super();
  earthquakes = cursor;

  quakeLocations = new ArrayList<GeoPoint>();
  refreshQuakeLocations();
  earthquakes.registerDataSetObserver(new DataSetObserver() {
    @Override
    public void onChanged() {
      refreshQuakeLocations();
    }
  });
}
```

5.4. Закончите создание EarthquakeOverlay, переопределив метод draw, который будет перебирать объекты GeoPoint из списка и рисовать для каждого из них отметку над местоположением землетрясения. В этом примере изображается красная окружность, но вы можете

это изменить, добавив дополнительную информацию (например, выбрать иной размер окружности в зависимости от силы землетрясения).

```
int rad = 5;

@Override
public void draw(Canvas canvas, MapView mapView, boolean shadow) {
  Projection projection = mapView.getProjection();

  // Создайте и настройте вашу кисть для рисования
  Paint paint = new Paint();
  paint.setARGB(250, 255, 0, 0);
  paint.setAntiAlias(true);
  paint.setFakeBoldText(true);

  if (shadow == false) {
    for (GeoPoint point : quakeLocations) {
      Point myPoint = new Point();
      projection.toPixels(point, myPoint);

      RectF oval = new RectF(myPoint.x-rad, myPoint.y-rad,
                             myPoint.x+rad, myPoint.y+rad);

      canvas.drawOval(oval, paint);
    }
  }
}
```

6. Перейдите к классу EarthquakeMap. Создайте внутри метода onCreate объект Cursor, возвращающий землетрясения, которые вы хотите отобразить на карте. Используйте Курсор для создания нового объекта EarthquakeOverlay, прежде чем добавлять экземпляр этого Наложения в список Наложений MapView.

```
Cursor earthquakeCursor;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.earthquake_map);

  String earthquakeURI = EarthquakeProvider.CONTENT_URI;
  earthquakeCursor = getContentResolver().query(earthquakeURI,
                     null, null, null,
                     null);

  MapView earthquakeMap = (MapView)findViewById(R.id.map_view);
  EarthquakeOverlay eo = new EarthquakeOverlay(earthquakeCursor);
  earthquakeMap.getOverlays().add(eo);
}
```

7. Наконец, переопределите обработчики onResume, чтобы получать новые землетрясения всякий раз, когда Активность становится видимой, onPause и onDestroy, чтобы оптимизировать использование ресурсов Курсора.

```
@Override
public void onResume() {
  earthquakeCursor.requery();
  super.onResume();
}

@Override
public void onPause() {
  earthquakeCursor.deactivate();
  super.onPause();
}

@Override
public void onDestroy() {
  earthquakeCursor.close();
  super.onDestroy();
}
```

8. Если вы запустите проект и выберете пункт Earthquake Map из главного меню, ваше приложение должно выглядеть, как на рис. 8.7.



Рис. 8.7.

Резюме

Геолокационные сервисы, геокодировщик и MapView применяются для создания интуитивно понятных приложений, работающих с географическими данными.

В этой главе вы познакомились с принципом работы геокодировщика, узнали об особенностях прямого и обратного геокодирования для перевода географических координат в адреса и обратно, использовали его для отслеживания передвижений и создания оповещений о близости нахождения.

Затем вы создали интерактивное приложение, использующее картографию. С помощью Наложений и Представлений нанесли аннотации на MapView с использованием двумерной графики, а также добавили отметки в виде объектов OverlayItem и Представлений (включая Группы представлений и разметку).

В главе 9 рассказывается, как работать в фоновом режиме. При ее прочтении вы познакомитесь с компонентом Service и научитесь переносить выполнение ресурсоемких задач в фоновые потоки. Чтобы взаимодействовать с пользователем, пока приложение скрыто, примените кратковременные сообщения Toast, а также объект Notification Manager для звонков, вибраций и мерцания светодиодов.

Глава 9
РАБОТА В ФОНОВОМ РЕЖИМЕ

Содержание главы

- Создание, запуск и остановка Сервисов.
- Привязка Сервисов к Активностям.
- Установка повышенного приоритета для Сервисов.
- Использование AsyncTask для управления фоновыми процессами.
- Создание фоновых потоков и использование обработчиков для синхронизации их с потоком GUI.
- Отображение уведомлений типа Toast.
- Использование NotificationManager для оповещения пользователя о событиях в приложении.
- Создание настойчивых и текущих уведомлений.
- Применения Сигнализации (Alarms) для планирования событий в приложении.

Android предоставляет класс Service для создания компонентов приложений, предназначенных специально для управления операциями и функциональностью, которые должны работать незаметно, без отображения пользовательского интерфейса.

Android дает Сервисам более высокий приоритет, чем бездействующим Активностям, поэтому вероятность того, что они будут завершены из-за нехватки ресурсов, заметно уменьшается. По сути, если система должна преждевременно завершить работу запущенного Сервиса, он может быть настроен таким образом, чтобы запускаться повторно, как только станет доступно достаточное количество ресурсов. В крайних случаях прекращение работы Сервиса (например, задержка при проигрывании музыки) будет заметно влиять на впечатления пользователя от приложения, и в подобных ситуациях приоритет Сервиса может быть повышен до уровня Активности, работающей на переднем плане.

Используя Сервисы, можете быть уверены, что ваши приложения продолжат работать и реагировать на события, даже если они в неактивном состоянии.

Для работы Сервисам не нужен отдельный графический интерфейс, как в случае с Активностями или Приемниками намерений, но они по-прежнему выполняются в главном потоке хода приложения. Чтобы повысить отзывчивость вашего приложения, нужно уметь переносить трудоемкие процессы (например, сетевые запросы) в фоновые потоки, используя классы Thread и AsyncTask.

Android предоставляет несколько способов, с помощью которых приложение может взаимодействовать с пользователем без наличия Активности. Вы узнаете, как использовать уведомления (обычные и типа Toast) для предупреждения и оповещения пользователя без необходимости прерывать работу активного приложения.

Уведомления типа Toast представляют собой механизм отображения кратковременных, немодальных диалоговых окон, которые выводят пользователю информацию, не забирая фокус с активного приложения. Вы научитесь отображать уведомления Toast из любого компонента приложения, чтобы слать пользователю ненавязчивые экранные сообщения.

Уведомления типа Toast — ненавязчивые и кратковременные уведомления класса Notification — предоставляют более надежный механизм для оповещения пользователей. Люди редко активно пользуются своим мобильным телефоном, чаще устройство находится вне поля зрения — в кармане или на столе, пока не начнет звонить, вибрировать или мерцать. Если пользователь пропустил эти уведомления, значки в строке состояния должны сигнализировать о том, что произошло какое-то событие. Все эти механизмы для привлечения внимания доступны вашим приложениям на Android с помощью уведомлений.

Сигнализация предоставляет механизм активизации Намерений в установленное время и за пределами жизненного цикла приложения. Вы узнаете, как использовать Сигнализацию для запуска Сервисов, открытия Активностей или трансляции Намерений, основываясь на текущем времени или времени, прошедшем с момента включения устройства. Сигнализация действует даже после того, как приложение, которому она принадлежит, закрылось, и может в случае необходимости вывести устройство из режима ожидания.

Знакомство с Сервисами

В отличие от Активностей, которые предоставляют пользователю насыщенный графический интерфейс, Сервисы работают в фоновом режиме, обновляя Источники данных, активизируя Намерения и показывая уведомления. Они идеально подходят для проведения постоянных или регулярных операций, а также для обработки событий даже тогда, когда Активности вашего приложения невидимы, работают в пассивном режиме или закрыты.

Сервисы запускаются, останавливаются и контролируются из различных компонентов приложения, включая другие Сервисы, Активности и Приемники

широковещательных намерений. Если ваше приложение выполняет задачи, которые не зависят от прямого взаимодействия с пользователем, Сервисы могут стать хорошим выбором.

Запущенные Сервисы всегда имеют больший приоритет, чем бездействующие или невидимые Активности, поэтому менее вероятно, что их работа завершится преждевременно при распределении ресурсов. Единственная причина, почему Android может досрочно остановить Сервис, — выделение дополнительных ресурсов для компонентов, работающих на переднем плане (как правило, для Активностей). Если такое случится, ваш Сервис автоматически перезапустится, когда будет достаточно доступных ресурсов.

Когда Сервис напрямую взаимодействует с пользователем (например, проигрывая музыку), может понадобиться повысить его приоритет до уровня Активностей, работающих на переднем плане. Это гарантия того, что Сервис завершится только в крайнем случае, но при этом снижается его доступность во время выполнения, мешая управлять ресурсами, что может испортить общее впечатление от приложения.

Приложения, которые регулярно обновляются, но очень редко или нерегулярно взаимодействуют с пользователем, можно назвать первыми кандидатами на реализацию в виде Сервисов. Проигрыватели MP3 и приложения, отслеживающие спортивные результаты, — примеры программ, которые должны постоянно работать и обновляться без необходимости отображать Активность.

Другие примеры можно найти в стандартном комплекте: Android предоставляет несколько Сервисов, включая LocationManager, MediaController и NotificationManager.

Создание Сервисов и управление ими

В следующих разделах вы узнаете, как создавать новые Сервисы, запускать и останавливать их, используя Намерения и метод startService. Позже научитесь привязывать Сервис к Активности, чтобы обеспечивать более богатый интерфейс для взаимодействия.

Создание Сервиса

Чтобы получить Сервис, создайте новый класс, который будет потомком класса Service. Вам необходимо переопределить обработчики onBind и onCreate, как показано в листинге 9.1.

Листинг 9.1. Каркас класса нового Сервиса

```
import android.app.Service;
import android.content.Intent;
```

Продолжение ⇗

Листинг 9.1 (продолжение)

```
import android.os.IBinder;

public class MyService extends Service {

  @Override
  public void onCreate() {
    // TODO: действия, которые будут выполняться при создании сервиса.
  }

  @Override
  public IBinder onBind(Intent intent) {
    // TODO: Заменить реализацией привязки сервиса.
    return null;
  }
}
```

В большинстве случаев, как правило, также необходимо переопределить метод onStartCommand. Он вызывается каждый раз, когда Сервис стартует с помощью метода startService, поэтому может быть выполнен несколько раз на протяжении работы. Вы должны убедиться, что ваш Сервис это предусматривает.

Обработчик onStartCommand заменяет событие onStart, которое использовалось в Android 2.0. В отличие от onStart он позволяет указать системе, каким образом обрабатывать перезапуски, если Сервис остановлен системой без явного вызова методов stopService или stopSelf.

Следующий фрагмент кода дополняет листинг 9.1 и демонстрирует каркас для переопределения обработчика onStartCommand. Обратите внимание, что от значения, которое он возвращает, зависит, как именно система прореагирует на перезапуск Сервиса в случае принудительного завершения его работы.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  // TODO Запустить поток в фоновом режиме для обработки.
  return Service.START_STICKY;
}
```

Сервисы запускаются в главном потоке приложения; это значит, что любые операции, выполняющиеся в обработчике onStartCommand, будут работать в контексте главного потока GUI. На практике при реализации Сервиса в методе onStartCommand создают и запускают новый поток, чтобы выполнять операции в фоновом режиме и останавливать Сервис, когда работа завершена (далее в этой главе вы узнаете, как создавать фоновые потоки и управлять ими).

Такой подход позволяет методу onStartCommand быстро завершить работу и дает возможность контролировать поведение Сервиса при его повторном запуске, используя одну из констант.

- START_STICKY. Описывает стандартное поведение. Похоже на то, как был реализован метод onStart в Android 2.0. Если вы вернете

это значение, обработчик onStartCommand будет вызываться при повторном запуске Сервиса после преждевременного завершения работы. Обратите внимание, что аргумент Intent, передаваемый в onStartCommand, получит значение null.

Данный режим обычно используется для Сервисов, которые сами обрабатывают свои состояния, явно стартуя и завершая свою работу при необходимости (с помощью startService и stopService). Это относится к Сервисам, которые проигрывают музыку или выполняют другие задачи в фоновом режиме.

- START_NOT_STICKY. Этот режим используется в Сервисах, которые запускаются для выполнения конкретных действий или команд. Как правило, такие Сервисы используют stopSelf для прекращения работы, как только команда выполнена.

  После преждевременного прекращения работы Сервисы, работающие в данном режиме, повторно запускаются только в том случае, если получат вызовы. Если с момента завершения работы Сервиса не был запущен метод startService, он остановится без вызова обработчика onStartCommand.

  Данный режим идеально подходит для Сервисов, которые обрабатывают конкретные запросы, особенно это касается регулярного выполнения заданных действий (например, обновления или сетевые запросы). Вместо того, чтобы перезапускать Сервис при нехватке ресурсов, часто более целесообразно позволить ему остановиться и повторить попытку запуска по прошествии запланированного интервала.

- START_REDELIVER_INTENT. В некоторых случаях нужно убедиться, что команды, которые вы посылаете Сервису, выполнены.

  Этот режим — комбинация предыдущих двух. Если система преждевременно завершила работу Сервиса, он запустится повторно, но только когда будет сделан явный запрос на запуск или если процесс завершился до вызова метода stopSelf.

  В последнем случае вызовется обработчик onStartCommand, он получит первоначальное Намерение, обработка которого не завершилась должным образом.

Обратите внимание, что при окончании всех операций каждый из этих режимов требует явной остановки Сервиса с помощью методов stopService или stopSelf. Оба эти метода подробнее описаны далее в этой главе.

ПРИМЕЧАНИЕ

В версиях, предшествующих Android SDK 2.0 (SDK API level 5), класс Service вызывал обработчик onStart, чтобы дать вам возможность выполнить какие-то действия при запуске Сервиса. Теперь же реализация обработчика onStart эквивалентна переопределению метода onStartCommand, возвращающего флаг START_STICKY.

Режим перезапуска, который вы указываете в качестве значения, возвращаемого методом onStartCommand, будет влиять на параметры, передаваемые при последующих вызовах.

Изначально Намерение выступает в качестве параметра, который передастся в метод startService при запуске Сервиса. После перезапуска системой он может иметь значение null (если установлен режим START_STICKY) или оригинальное (если установлен флаг START_REDELIVER_INTENT).

Параметр flag может помочь узнать, как именно был запущен Сервис. В частности, вы можете использовать фрагмент кода, показанный в листинге 9.2, чтобы определить, какая из констант соответствует параметру flag:

- START_FLAG_REDELIVERY — указывает на то, что параметр Intent повторно передан при принудительном завершении работы Сервиса перед явным вызовом метода stopSelf;
- START_FLAG_RETRY — указывает на то, что Сервис повторно запущен после непредвиденного завершения работы; передается в том случае, если ранее Сервис работал в режиме START_STICKY.

Листинг 9.2. Определение причины запуска Сервиса системой

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  if ((flags & START_FLAG_RETRY) == 0) {
    // TODO Если это повторный запуск, выполнить какие-то действия.
  }
  else {
    // TODO Альтернативные действия в фоновом режиме.
  }
  return Service.START_STICKY;
}
```

Регистрация Сервиса в манифесте

Создав новый Сервис, нужно зарегистрировать его в манифесте приложения.

Сделайте это, добавив тег service в узел application. Используйте атрибут requires-permission, чтобы и другие приложения имели доступ к вашему Сервису.

Ниже показан тег service, который нужно добавить для каркаса вашего Сервиса, созданного ранее:

```
<service android:enabled="true" android:name=".MyService"/>
```

Завершение работы Сервиса по собственной инициативе

Как только Сервис завершил выполнение тех действий, для которых он запускался, вы должны вызвать метод stopSelf либо без передачи параметра,

чтобы ускорить остановку работы, либо передав значение startId, чтобы убедиться, что задачи выполнены для всех экземпляров, запущенных с помощью вызова startService, как показано в следующем фрагменте:

```
stopSelf(startId);
```

Явная остановка Сервиса по завершении необходимых задач позволяет системе получать обратно ресурсы, которые в ином случае оставались бы заняты. Поскольку приоритет Сервисов повышенный, система, как правило, не завершает их работу, поэтому ее окончание по собственной инициативе может существенно улучшить эффективность использования ресурсов вашим приложением.

Запуск, управление и взаимодействие с Сервисами

Чтобы запустить Сервис, вызовите метод startService. Вы можете произвести запуск неявно, в итоге зарегистрируется соответствующий Приемник намерений, или сделать это явно, указав класс Сервиса. Если Сервис требует полномочий, которыми ваше приложение не располагает, вызов метода startService приведет к выбросу исключения типа SecurityException.

В обоих случаях можно передавать значения в обработчик onStart, принадлежащий объекту Service, с помощью дополнительных параметров для Намерения. В листинге 9.3 демонстрируются оба способа запуска Сервиса.

Листинг 9.3. Запуск Сервиса

```
// Неявный запуск Сервиса
Intent myIntent = new Intent(MyService.ORDER_PIZZA);
myIntent.putExtra("TOPPING", "Margherita");
startService(myIntent);

// Явный запуск Сервиса
startService(new Intent(this, MyService.class));
```

ПРИМЕЧАНИЕ

Чтобы использовать этот пример, необходимо добавить константу MY_ACTION к классу MyService и использовать Фильтр намерений для регистрации Сервиса в качестве поставщика поля MY_ACTION.

Для остановки работы используйте метод stopService, передавая ему объект Intent, определяющий нужный Сервис. В листинге 9.4 Сервис сперва запускается и останавливается явным образом, а затем делается то же самое, но с указанием имени компонента, возвращаемого методом startService.

Листинг 9.4. Остановка Сервиса

```
ComponentName service = startService(new Intent(this, BaseballWatch.
class));
```

Продолжение ⇗

Листинг 9.4 (продолжение)

```
// Остановите Сервис, используя его имя.
stopService(new Intent(this, service.getClass()));
// Остановите Сервис явно.
try {
  Class serviceClass = Class.forName(service.getClassName());
  stopService(new Intent(this, serviceClass));
} catch (ClassNotFoundException e) {}
```

Если метод startService вызывается для Сервиса, который уже работает, обработчик onStartCommand, принадлежащий объекту Service, будет вызван повторно. Вызовы startService не накапливаются, поэтому единственный вызов метода stopService завершит работу Сервиса, неважно, сколько раз производился вызов startService.

Пример Сервиса для отслеживания землетрясений

В этой главе вы модифицируете приложение Earthquake, работу над которым начали в главе 5 и продолжили в главах 6, 7 и 8. В данном примере[1] перенесете процесс обновления списка землетрясений и обработку полученной информации в отдельный компонент типа Service.

ПРИМЕЧАНИЕ

Позже в данной главе вы снабдите этот Сервис дополнительной функциональностью, начав с переноса сетевых запросов и обработки XML в фоновый поток. Далее задействуете уведомления Toast и Notification для оповещения пользователя о новых землетрясениях.

1. Начните с создания нового класса EarthquakeService, наследующего Service.

```
package com.paad.earthquake;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import java.util.Timer;
import java.util.TimerTask;

public class EarthquakeService extends Service {
  @Override
  public void onCreate() {
    // TODO: Инициализируйте переменные, получите ссылки на элементы GUI
  }

  @Override
  public IBinder onBind(Intent intent) {
    return null;
  }
}
```

[1] Все фрагменты кода в этом примере — часть проекта Earthquake из главы 9, их можно загрузить с сайта Wrox.com.

2. Включите этот Сервис в манифест, добавив новый тег `service` внутрь узла application.

```
<service android:enabled="true" android:name=".EarthquakeService"/>
```

3. Переместите методы refreshEarthquakes и addNewQuake из Активности Earthquake в EarthquakeService. Вы должны убрать вызовы методов addQuakeToArray и loadQuakesFromProvider (но сами методы не трогайте, они еще понадобятся). Удалите из EarthquakeService все ссылки на массив ArrayList, принадлежащий Активности Earthquake.

```
private void addNewQuake(Quake _quake) {
  ContentResolver cr = getContentResolver();
  // Создайте оператор WHERE, чтобы удостовериться, что у нас в источнике
  // еще нет данного землетрясения.
  String w = EarthquakeProvider.KEY_DATE + " = " +
             _quake.getDate().getTime();

  // Если землетрясение новое, вставьте его в источник.
  Cursor c = cr.query(EarthquakeProvider.CONTENT_URI,
                      null, w, null, null);
  if (c.getCount()==0){
    ContentValues values = new ContentValues();

    values.put(EarthquakeProvider.KEY_DATE,
    _quake.getDate().getTime());
    values.put(EarthquakeProvider.KEY_DETAILS, _quake.getDetails());

    double lat = _quake.getLocation().getLatitude();
    double lng = _quake.getLocation().getLongitude();
    values.put(EarthquakeProvider.KEY_LOCATION_LAT, lat);
    values.put(EarthquakeProvider.KEY_LOCATION_LNG, lng);
    values.put(EarthquakeProvider.KEY_LINK, _quake.getLink());
    values.put(EarthquakeProvider.KEY_MAGNITUDE, _quake.getMagnitude());

    cr.insert(EarthquakeProvider.CONTENT_URI, values);
  }
  c.close();
}

private void refreshEarthquakes() {
  // Получите код XML
  URL url;
  try {
    String quakeFeed = getString(R.string.quake_feed);
    url = new URL(quakeFeed);

    URLConnection connection;
    connection = url.openConnection();

    HttpURLConnection httpConnection =
      (HttpURLConnection)connection;
    int responseCode = httpConnection.getResponseCode();

    if (responseCode == HttpURLConnection.HTTP_OK) {
```

```
InputStream in = httpConnection.getInputStream();

DocumentBuilderFactory dbf =
  DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();

// Обработайте ленту с землетрясениями.
Document dom = db.parse(in);
Element docEle = dom.getDocumentElement();

// Получите список каждого элемента землетрясения.
NodeList nl = docEle.getElementsByTagName("entry");
if (nl != null && nl.getLength() > 0) {
  for (int i = 0 ; i < nl.getLength(); i++) {
    Element entry = (Element)nl.item(i);
    Element title;
    title =
      (Element)entry.getElementsByTagName("title").item(0);
    Element g =
      (Element)entry.getElementsByTagName("georss:point").item(0);
    Element when =
      (Element)entry.getElementsByTagName("updated").item(0);
    Element link =
      (Element)entry.getElementsByTagName("link").item(0);

    String details = title.getFirstChild().getNodeValue();
    String hostname = "http://earthquake.usgs.gov";
    String linkString = hostname + link.getAttribute("href");

    String point = g.getFirstChild().getNodeValue();
    String dt = when.getFirstChild().getNodeValue();
    SimpleDateFormat sdf;
    sdf = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss'Z'");
    Date qdate = new GregorianCalendar(0,0,0).getTime();
    try {
      qdate = sdf.parse(dt);
    } catch (ParseException e) {
      e.printStackTrace();
    }

    String[] location = point.split(" ");
    Location l = new Location("parsed");
    l.setLatitude(Double.parseDouble(location[0]));
    l.setLongitude(Double.parseDouble(location[1]));

    String magnitudeString = details.split(" ")[1];
    int end = magnitudeString.length()-1;
    double magnitude =
      Double.parseDouble(magnitudeString.substring(0, end));

    details = details.split(",")[1].trim();

    Quake quake = new Quake(qdate, details, l, magnitude,
                            linkString);

    // Обработайте только что полученное землетрясение.
```

```
          addNewQuake(quake);
        }
      }
    }
  } catch (MalformedURLException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  } catch (ParserConfigurationException e) {
    e.printStackTrace();
  } catch (SAXException e) {
    e.printStackTrace();
  }
  finally {
  }
}
```

4. Внутри Активности Earthquake создайте новый метод refreshEarthquakes. Он должен явным образом запускать Сервис EarthquakeService.

```
private void refreshEarthquakes() {
  startService(new Intent(this, EarthquakeService.class));
}
```

5. Вернитесь к EarthquakeService. Переопределите методы onStartCommand и onCreate для поддержки нового объекта Timer, который будет использован для обновления списка с землетрясениями. onStartCommand должен возвращать START_STICKY, так как задействуется таймер для запуска нескольких обновлений. По большому счету, это не совсем правильно, работа объекта Timer должна быть перемещена в фоновый поток и инициироваться с помощью Сигнализации. Позже в этой главе вы научитесь все это делать.

   Используйте объект SharedPreference, созданный в главе 6, чтобы определять, должна ли информация о землетрясениях обновляться регулярно.

```
private Timer updateTimer;
private float minimumMagnitude;

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  // Получите Общие настройки
  SharedPreferences prefs =
    getSharedPreferences(Preferences.USER_PREFERENCE,
                         Activity.MODE_PRIVATE);

  int minMagIndex = prefs.getInt(Preferences.PREF_MIN_MAG, 0);
  if (minMagIndex < 0)
    minMagIndex = 0;

  int freqIndex = prefs.getInt(Preferences.PREF_UPDATE_FREQ, 0);
```

```
  if (freqIndex < 0)
    freqIndex = 0;

  boolean autoUpdate =
    prefs.getBoolean(Preferences.PREF_AUTO_UPDATE, false);

  Resources r = getResources();
  int[] minMagValues = r.getIntArray(R.array.magnitude);
  int[] freqValues = r.getIntArray(R.array.update_freq_values);

  minimumMagnitude = minMagValues[minMagIndex];
  int updateFreq = freqValues[freqIndex];

  updateTimer.cancel();
  if (autoUpdate) {
    updateTimer = new Timer("earthquakeUpdates");
    updateTimer.scheduleAtFixedRate(doRefresh, 0,
      updateFreq*60*1000);
  }
  else
    refreshEarthquakes();

  return Service.START_STICKY;
};

private TimerTask doRefresh = new TimerTask() {
  public void run() {
    refreshEarthquakes();
  }
};

@Override
public void onCreate() {
  updateTimer = new Timer("earthquakeUpdates");
}
```

6. Теперь EarthquakeService будет обновлять источник для землетрясений каждый раз, когда возникнет необходимость (как и по автоматическому расписанию, если таковое было создано). Обновления пока еще не возвращаются обратно в компонент ListView Активности Earthquake или в Активность MapActivity.

   Для оповещения этих компонентов и любых других заинтересованных приложений о новых данных отредактируйте EarthquakeService, чтобы отсылать новое Намерение при каждом добавлении элемента в источник землетрясений.

   6.1. Отредактируйте метод addNewQuake, добавив в него вызов нового метода announceNewQuake.

```
public static final String NEW_EARTHQUAKE_FOUND = "New_Earthquake_Found";

private void addNewQuake(Quake _quake) {
  ContentResolver cr = getContentResolver();
  // Создайте оператор WHERE, чтобы удостовериться, что у нас в источнике
  // еще нет данного землетрясения.
    String w = EarthquakeProvider.KEY_DATE +
```

```
            " = " + _quake.getDate().getTime();

  // Если землетрясение новое, вставьте его в источник.
  Cursor c = cr.query(EarthquakeProvider.CONTENT_URI,
                      null, w, null, null);
  if (c.getCount()==0){
    ContentValues values = new ContentValues();

    values.put(EarthquakeProvider.KEY_DATE, _quake.getDate().getTime());
    values.put(EarthquakeProvider.KEY_DETAILS, _quake.getDetails());

    double lat = _quake.getLocation().getLatitude();
    double lng = _quake.getLocation().getLongitude();
    values.put(EarthquakeProvider.KEY_LOCATION_LAT, lat);
    values.put(EarthquakeProvider.KEY_LOCATION_LNG, lng);
    values.put(EarthquakeProvider.KEY_LINK, _quake.getLink());
    values.put(EarthquakeProvider.KEY_MAGNITUDE,
               _quake.getMagnitude());

    cr.insert(EarthquakeProvider.CONTENT_URI, values);
    announceNewQuake(_quake);
  }
  c.close();
}

private void announceNewQuake(Quake quake) {
}
```

6.2. Внутри метода announceNewQuake отсылайте Намерение каждый раз, когда обнаруживается новое землетрясение.

```
private void announceNewQuake(Quake quake) {
  Intent intent = new Intent(NEW_EARTHQUAKE_FOUND);
  intent.putExtra("date", quake.getDate().getTime());
  intent.putExtra("details", quake.getDetails());
  intent.putExtra("longitude", quake.getLocation().getLongitude());
  intent.putExtra("latitude", quake.getLocation().getLatitude());
  intent.putExtra("magnitude", quake.getMagnitude());

  sendBroadcast(intent);
}
```

7. На этом реализация EarthquakeService завершена. Однако вам еще необходимо отредактировать две Активности, чтобы следить за тем, как Сервис шлет Намерение и обновлять при этом отображаемые данные.

7.1. Внутри Активности Earthquake создайте новый внутренний класс EarthquakeReceiver, который наследует BroadcastReceiver. Переопределите обработчик onReceive, чтобы вызвать метод loadFromProviders для обновления списка и массива с землетрясениями.

```
public class EarthquakeReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    loadQuakesFromProvider();
  }
}
```

7.2. Переопределите метод onResume, чтобы зарегистрировать новый объект Receiver и обновить содержимое элемента ListView, когда Активность перейдет на первый план. Переопределите обработчик onPause для отмены регистрации, когда Активность перейдет в режим бездействия.

```
EarthquakeReceiver receiver;

@Override
public void onResume() {
  IntentFilter filter;
  filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
  receiver = new EarthquakeReceiver();
  registerReceiver(receiver, filter);

  loadQuakesFromProvider();
  super.onResume();
}

@Override
public void onPause() {
  unregisterReceiver(receiver);
  super.onPause();
}
```

7.3. Повторите то же самое для Активности EarthquakeMap, но на этот раз при получении Намерения вызывайте метод requery для результирующего Курсора перед обновлением MapView.

```
EarthquakeReceiver receiver;

@Override
public void onResume() {
  earthquakeCursor.requery();

  IntentFilter filter;
  filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
  receiver = new EarthquakeReceiver();
  registerReceiver(receiver, filter);

  super.onResume();
}

@Override
public void onPause() {
  earthquakeCursor.deactivate();
  super.onPause();
}

public class EarthquakeReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    earthquakeCursor.requery();
```

```
    MapView earthquakeMap = (MapView)findViewById(R.id.map_view);
    earthquakeMap.invalidate();
  }
}
```

Теперь, когда загрузится Активность Earthquake, запустится Сервис EarthquakeService и будет продолжать работу, обновляя в фоновом режиме Источник данных с информацией о землетрясениях, даже если Активность приостановится или закроется.

ПРИМЕЧАНИЕ

В этой главе вы продолжите дополнять и улучшать Earthquake Service, сперва используя уведомления типа Toast, а затем Notification и Сигнализацию.

На данном этапе обработка информации о землетрясениях происходит внутри Сервиса, но он все еще выполняется в контексте главного потока GUI. Позже в этой главе вы узнаете, как переместить трудоемкие операции в фоновые потоки, чтобы улучшить производительность и избежать сообщений о необходимости закрытия приложения.

Кроме того, Сервис постоянно работает, отбирая ценные ресурсы. В следующих разделах показано, как заменить таймер на Сигнализацию.

Привязка Активностей к Сервисам

Если Активность связана с Сервисом, она может хранить у себя ссылку на экземпляр объекта Service, что позволяет вызывать методы работающего Сервиса, как в случае с экземпляром любого другого класса.

Подобного рода привязка доступна для Активностей, которые выигрывают от наличия более основательного интерфейса для работы с Сервисом. Чтобы поддерживать связь с Сервисом, реализуйте метод onBind, как показано в листинге 9.5.

Листинг 9.5. Реализация привязки к Сервису

```
private final IBinder binder = new MyBinder();

@Override
  public IBinder onBind(Intent intent) {
  return binder;
}

public class MyBinder extends Binder {
  MyService getService() {
    return MyService.this;
  }
}
```

Связь между Сервисом и Активностью представлена классом ServiceConnection. Вам нужно реализовать новый класс ServiceConnection, переопределив методы onServiceConnected и onServiceDisconnected, чтобы получить ссылку на экземпляр Сервиса при установлении связи, как показано в листинге 9.6.

Листинг 9.6. Привязка к Сервису

```
// Ссылка на Сервис
private MyService serviceBinder;

// Обработка связи между Сервисом и Активностью
private ServiceConnection mConnection = new ServiceConnection() {
  public void onServiceConnected(ComponentName className, IBinder
service) {
    // Вызывается при установлении связи.
    serviceBinder = ((MyService.MyBinder)service).getService();
  }

  public void onServiceDisconnected(ComponentName className) {
    // Вызывается при неожиданном разрыве связи.
    serviceBinder = null;
  }
};
```

Для привязки вызовите метод bindService, передав в качестве параметров Намерение (явное или неявное), которое в свою очередь получит нужный Сервис, а также реализацию ServiceConnection, как показано в следующем фрагменте, дополняющем листинг 9.6:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  // Свяжите Активность с Сервисом
  Intent bindIntent = new Intent(MyActivity.this, MyService.class);
  bindService(bindIntent, mConnection, Context.BIND_AUTO_CREATE);
}
```

Как только Сервис будет привязан, все его публичные методы и свойства станут доступными через объект serviceBinder, полученный в обработчике onServiceConnected.

Приложения в Android, как правило, не имеют общей памяти, но в некоторых случаях вашему приложению может потребоваться связаться с Сервисами, работающими в процессах других приложений.

Вы можете установить взаимодействие с Сервисами, функционирующими в других процессах, с помощью передачи Намерений или через дополнительные параметры. Параметр Bundle в Намерении используется для запуска Сервиса. Если вам нужна более тесная связь, можете сделать Сервис доступным для привязки вне зависимости от приложения, используя AIDL. AIDL определяет интерфейс Сервиса на уровне операционной системы, по-

зволяя Android передавать объекты между процессами. Более подробно об AIDL рассказывается в главе 15.

Назначение приоритетов для фоновых Сервисов

Как вы уже узнали из главы 3, Android применяет динамический подход к управлению ресурсами, что может привести к преждевременному завершению работы ваших приложений, Активностей и Сервисов практически без предупреждения.

При определении, какие приложения и компоненты приложений должны быть закрыты, Android присваивает запущенным Сервисам второй по значимости приоритет. Только Активности, работающие на переднем плане, имеют более высокий приоритет с точки зрения системных ресурсов.

В крайних случаях, когда ваш Сервис напрямую взаимодействует с пользователем, можно поднять его приоритет до наивысшего, эквивалентного Активностям, работающим на переднем плане. Делается это с помощью метода startForeground.

Ожидается, что Сервис, работающий на переднем плане, будет напрямую взаимодействовать с пользователем (например, проигрывая музыку). В связи с этим клиент всегда должен знать о работе таких Сервисов. Чтобы это обеспечить, вызов startForeground должен сопровождаться появлением текущих уведомлений (которые более подробно описываются далее в этой главе), как показано в листинге 9.7. Эти уведомления будут работать до тех пор, пока Сервис на переднем плане.

ПРИМЕЧАНИЕ

Перемещая Сервис на передний план, вы фактически запрещаете системе завершать его работу для освобождения ресурсов. При наличии нескольких таких Сервисов, задействованных одновременно, системе чрезвычайно сложно восстанавливать ресурсы при их нехватке.

Используйте данный подход только тогда, когда это необходимо для правильной работы Сервиса. И даже в таких случаях сохраняйте повышенный приоритет ровно столько, сколько это нужно.

Листинг 9.7. Повышение приоритета для Сервиса

```
int NOTIFICATION_ID = 1;
Intent intent = new Intent(this, MyActivity.class);
PendingIntent pi = PendingIntent.getActivity(this, 1, intent, 0));

Notification notification = new Notification(R.drawable.icon,
  "Running in the Foreground", System.currentTimeMillis());
notification.setLatestEventInfo(this, "Title", "Text", pi);

notification.flags = notification.flags |
                     Notification.FLAG_ONGOING_EVENT;

startForeground(NOTIFICATION_ID, notification);
```

В листинге 9.7 используется метод setLatestEventInfo для обновления содержимого уведомлений с помощью разметки статусной строки по умолчанию. Позже в этой главе вы научитесь указывать собственную разметку для уведомлений. Используя данный подход, можете предоставить пользователю более детальную информацию о работающем Сервисе.

Когда Сервис больше не нуждается в повышенном приоритете, можете вернуть его в фоновый режим и, если необходимо, убрать уведомления с помощью метода stopForeground, как показано в листинге 9.8. Уведомления будут отменены автоматически при остановке или завершении работы Сервиса.

Листинг 9.8. Возвращение Сервиса в фоновый режим

```
// Переход в фоновый режим и удаление уведомлений
stopForeground(true);
```

ПРИМЕЧАНИЕ

В версиях Android, предшествовавших 2.0, была возможность вывести Сервис на передний план с помощью метода setForeground. Этот метод сейчас считается устаревшим, его вызов ни к чему не приведет.

Использование фоновых потоков

Чтобы быть уверенным, что ваше приложение не теряет отзывчивости, хорошим решением станет перемещение всех медленных, трудоемких операций из главного потока приложения в дочерний.

ПРИМЕЧАНИЕ

Все компоненты приложения в Android, включая Активности, Сервисы и Приемники широковещательных намерений, начинают работу в главном потоке приложения. В результате трудоемкие операции в любом из этих компонентов блокируют все остальные части приложения, включая Сервисы и Активности на переднем плане.

Android предоставляет два способа перемещения функциональности в фоновый режим. Класс AsyncTask позволяет определить операции, которые будут выполняться в фоне, вы также будете иметь доступ к обработчику событий, что позволит отслеживать прогресс выполнения задач и выводить результаты в контексте главного графического потока.

Кроме того, вы можете реализовать ваш собственный класс, наследованный от Thread, используя объект Handler для синхронизации с потоком

GUI перед обновлением пользовательского интерфейса. В этом разделе описываются оба подхода.

Применение фоновых потоков — необходимое условие, если вы хотите избежать появления диалогового окна для принудительного закрытия приложения, которое уже описывалось в главе 2. Когда Активность в Android на протяжении 5 секунд не отвечает на события пользовательского ввода (например, нажатие кнопки) или Приемник широковещательных намерений не завершает работу обработчика onReceive в течение 10 секунд, считается, что приложение зависло.

Подобные ситуации не просто недопустимы, даже риск их возникновения нежелателен. Используйте фоновые потоки для всех трудоемких операций, включая работу с файлами, сетевые запросы, транзакции в базах данных и сложные вычисления.

Использование AsyncTask для запуска асинхронных задач

Класс AsyncTask предлагает простой и удобный механизм для перемещения трудоемких операций в фоновый поток. Благодаря ему вы получаете удобство синхронизации обработчиков событий с графическим потоком, что позволяет обновлять Представления и другие элементы пользовательского интерфейса для отчета о ходе выполнения задачи или для вывода результатов, когда задача завершена.

AsyncTask создает, синхронизирует потоки, а также управляет ими, что позволяет создавать асинхронные задачи, состоящие из операций, выполняющихся в фоновом режиме, и обновлять пользовательский интерфейс по их завершении.

Создание новой асинхронной задачи

Чтобы создать новую асинхронную задачу, понадобится наследовать класс AsyncTask, как показано на примере каркаса в листинге 9.9. Ваша реализация должна предусматривать классы для объектов, которые будут переданы в качестве параметров методу execute, для переменных, что станут использоваться для оповещения о ходе выполнения, а также для переменных, где будет храниться результат. Формат такой записи следующий:

```
AsyncTask<[Input Parameter Type], [Progress Report Type], [Result Type]>
```

Если не нужно или вы не хотите принимать параметры, обновлять информацию о ходе выполнения или выводить конечный результат, просто укажите тип Void во всех трех случаях.

Листинг 9.9. Каркас реализации AsyncTask, в котором используются строковой параметр и два целочисленных значения, нужных для оповещения о выполнении работы и о конечном результате

```
private class MyAsyncTask extends AsyncTask<String, Integer, Integer> {
  @Override
  protected void onProgressUpdate(Integer... progress) {
    // [... Обновите индикатор хода выполнения, уведомления или другой
элемент пользовательского интерфейса ...]
  }

  @Override
  protected void onPostExecute(Integer... result) {
    // [... Сообщите о результате через обновление пользовательского
интерфейса, диалоговое окно или уведомление ...]
  }

  @Override
  protected Integer doInBackground(String... parameter) {
    int myProgress = 0;

    // [... Выполните задачу в фоновом режиме, обновите переменную
myProgress...]
    PublishProgress(myProgress)
    // [... Продолжение выполнения фоновой задачи ...]

    // Верните значение, ранее переданное в метод onPostExecute
    return result;
  }
}
```

Как показано в листинге 9.9, дочерний класс должен реализовать несколько обработчиков событий.

- doInBackground. Принимает набор параметров тех типов, которые определены в реализации вашего класса. Этот метод выполняется в фоновом потоке, поэтому в нем не должно быть никакого взаимодействия с элементами пользовательского интерфейса. Размещайте здесь трудоемкий код, используя метод publishProgress, который позволит обработчику onProgressUpdate передавать изменения в пользовательский интерфейс. Когда фоновая задача завершена, данный метод возвращает конечный результат для обработчика onPostExecute, который сообщит о нем в поток пользовательского интерфейса.
- onProgressUpdate. Переопределите этот обработчик для публикации промежуточных обновлений в пользовательский интерфейс. При вызове он синхронизируется с потоком GUI, поэтому в нем вы можете безопасно изменять элементы пользовательского интерфейса.
- onPostExecute. Когда метод doInBackground завершает работу, конечный результат передается в этот обработчик событий. Используйте его для обновления пользовательского интерфейса, как только ваша фоновая задача завершена. Данный обработчик при вызове синхро-

низируется с потоком GUI, поэтому внутри него вы можете безопасно изменять элементы пользовательского интерфейса.

Запуск асинхронной задачи

Поскольку вы уже реализовали асинхронную задачу, запустите ее, создав новый экземпляр и вызвав метод execute, как показано в листинге 9.10. Вы можете передать параметры, тип каждого из которых был указан при реализации класса.

Листинг 9.10. Запуск асинхронной задачи

```
new MyAsyncTask().execute("inputString1", "inputString2");
```

ПРИМЕЧАНИЕ

Каждый экземпляр класса AsyncTask может быть запущен всего один раз. Попытка повторного вызова метода execute приведет к выбросу исключения.

Перенос Сервиса Earthquake в фоновый поток с помощью AsyncTask

В следующем примере[1] показывается, как с помощью AsyncTask перенести сетевые запросы и обработку XML, которые размещены в EarthquakeService, в фоновый поток.

1. Создайте новую реализацию AsyncTask, дайте ей название EarthquakeLookupTask, указав Void в качестве типа для входных параметров и конечного результата, а также Quake для сообщений о ходе выполнения операций. Добавьте заглушки, которые переопределяют методы doInBackground, onProgressUpdate и onPostExecute.

```
private class EarthquakeLookupTask extends AsyncTask<Void, Quake, Void> {
  @Override
  protected Void doInBackground(Void... params) {
    return null;
  }

  @Override
  protected void onProgressUpdate(Quake... values) {
    super.onProgressUpdate(values);
  }

  @Override
  protected void onPostExecute(Void result) {
    super.onPostExecute(result);
  }
}
```

[1] Все фрагменты кода в этом примере — часть проекта Earthquake 2 из главы 9, их можно загрузить с сайта Wrox.com.

2. Перенесите весь существующий код из метода refreshEarthquakes в новый обработчик doInBackground. Добавьте новый вызов метода publishProgress, который будет принимать последний извлеченный объект Quake при обработке нового землетрясения. После завершения обработки верните null.

```
@Override
protected Void doInBackground(Void... params) {
  [ ... ранее написанная обработка XML ... ]

  // Обработайте только что обнаруженное землетрясение
  addNewQuake(quake);
  publishProgress(quake);

  [ ... ранее написанный обработчик исключительных ситуаций ... ]

  return null;
}
```

3. Отредактируйте пока что пустой метод refreshEarthquakes. Он должен создавать и запускать новую задачу EarthquakeLookupTask. Сперва проверьте, не была ли ранее запущена другая асинхронная задача. Чтобы избежать накладки, необходимо инициировать обновление только в том случае, если оно еще не начало выполняться.

```
EarthquakeLookupTask lastLookup = null;

private void refreshEarthquakes() {
  if (lastLookup == null ||
      lastLookup.getStatus().equals(AsyncTask.Status.FINISHED)) {
    lastLookup = new EarthquakeLookupTask();
    lastLookup.execute((Void[])null);
  }
}
```

Создание потоков вручную и синхронизация с потоком GUI

Хотя использование AsyncTask — хорошее решение, случается, что для работы в фоновом режиме приходится создавать собственные потоки и управлять ими.

В этом разделе вы узнаете, как создавать и запускать новые объекты Thread, синхронизировать их работу с потоком GUI перед обновлением пользовательского интерфейса.

Создание нового потока

Вы можете создавать дочерние потоки и управлять ими с помощью класса Handler, предоставляемого Android, а также классов, доступных в пространстве имен java.lang.Thread. В листинге 9.11 показан простой каркас для переноса операций в дочерний поток.

Листинг 9.11. Перенос операций в фоновый поток

```
// Этот метод вызывается из главного потока GUI.
private void mainProcessing() {
  // Здесь трудоемкие задачи переносятся в дочерний поток.
  Thread thread = new Thread(null, doBackgroundThreadProcessing,
                             "Background");
  thread.start();
}

// Объект Runnable, который запускает метод для выполнения задач
// в фоновом режиме.
private Runnable doBackgroundThreadProcessing = new Runnable() {
  public void run() {
    backgroundThreadProcessing();
  }
};

// Метод, который выполняет какие-то действия в фоновом режиме.
private void backgroundThreadProcessing() {
  [ ... Трудоемкие операции ... ]
}
```

Использование объекта Handler для выполнения операций с GUI

Используя фоновые потоки в графической среде, важно синхронизировать их с главным потоком приложения (GUI), прежде чем создавать или изменять графические элементы.

Уведомления и Намерения, размещенные внутри компонентов вашего приложения, всегда принимаются и обрабатываются в графическом потоке. Во всех остальных случаях операции, явным образом взаимодействующие с объектами, созданными в потоке GUI (например, Представлениями), или отображающие сообщения (такие как объекты Toast), должны вызываться в главном потоке.

При работе в контексте Активности вы также можете использовать метод runOnUiThread, который принудительно выполняет задачу в потоке текущей Активности, как показано в листинге 9.12.

Листинг 9.12. Синхронизация с потоком Активности

```
runOnUiThread(new Runnable() {
  public void run() {
    // TODO Обновить Представление.
  }
});
```

При иных обстоятельствах (например, отображении Toast или Notification) можете использовать класс Handler для перемещения методов в поток, в котором сам Handler и создан.

С помощью класса Handler (точнее, его метода post) можно обновлять пользовательский интерфейс из фонового потока. В листинге 9.13 показана схема использования объекта Handler для обновления потока GUI.

Листинг 9.13. Использование Handler для синхронизации с графическим потоком

```
// Инициализируйте Handler в главном потоке.
private Handler handler = new Handler();

private void mainProcessing() {
  Thread thread = new Thread(null, doBackgroundThreadProcessing,
                             "Background");
  thread.start();
}

private Runnable doBackgroundThreadProcessing = new Runnable() {
  public void run() {
    backgroundThreadProcessing();
  }
};

// Метод, который выполняет какие-то задачи в фоновом режиме.
private void backgroundThreadProcessing() {
  [ ... Трудоемкие операции ... ]
  handler.post(doUpdateGUI);
}

// Объект Runnable, который запускает метод для обновления GUI.
private Runnable doUpdateGUI = new Runnable() {
  public void run() {
    updateGUI();
  }
};

private void updateGUI() {
  [ ... Открытие диалога или изменение графического элемента ... ]
}
```

Класс Handler позволяет вам задерживать выполнение обновлений или проводить их в заданное время. Делается это с помощью методов postDelayed и postAtTime соответственно.

Вывод уведомлений типа Toast

Toast — диалоговое окно, которое остается видимым всего несколько секунд перед тем, как исчезнуть. Этот вид уведомлений не перебирает на себя фокус и не считается модальным, поэтому он не прерывает работу активного приложения.

Toast идеально подходит для оповещения пользователей о событиях, не заставляя их при этом открывать Активность или читать текст уведомления. Toast предоставляет безупречный механизм для уведомления пользо-

вателя о событиях, происходящих в фоновых Сервисах, не приостанавливая при этом работу активных приложений.

Класс Toast включает статический метод makeText, который создает стандартное окно с уведомлением. Чтобы создать новое уведомление типа Toast, передайте методу makeText объект Context, принадлежащий приложению, текст сообщения, который нужно отобразить, а также время его отображения (LENGTH_SHORT или LENGTH_LONG). Как только объект Toast создан, отобразите его на экране, вызвав метод show, как показано в листинге 9.14.

Листинг 9.14. Отображение уведомления типа Toast

```
Context context = getApplicationContext();
String msg = "To health and happiness!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, msg, duration);
toast.show();
```

Уведомление типа Toast показано на рис. 9.1. Оно будет отображаться на экране около двух секунд, прежде чем исчезнет. Приложение, поверх которого оно выводится, остается полностью отзывчивым и может взаимодействовать с пользователем, даже когда Toast видим.

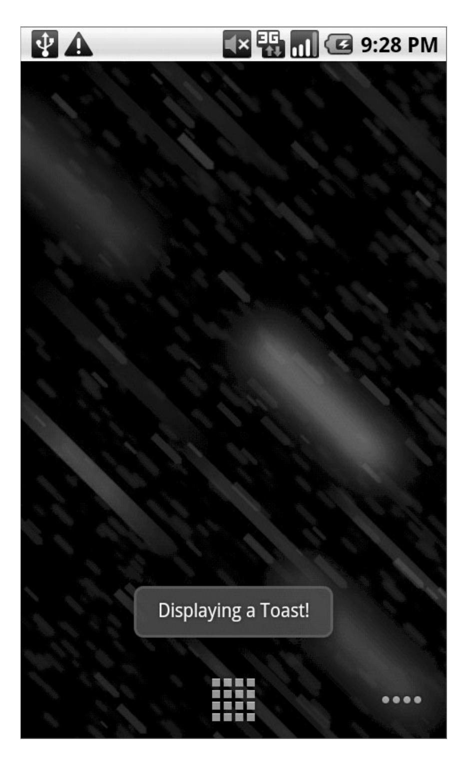


Рис. 9.1.

Настройка уведомлений типа Toast

Стандартного окна с текстовым сообщением, которое предлагает Toast, во многих ситуациях достаточно. Но иногда нужно будет настроить его внешний вид и позицию на экране. Вы можете изменить объект Toast, указав для него позицию на экране, где он будет размещен, а также альтернативное Представление или разметку.

В листинге 9.15 показано, как выровнять Toast по нижней границе экрана, используя метод setGravity.

Листинг 9.15. Настройка Toast

```
Context context = getApplicationContext();
String msg = "To the bride and groom!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, msg, duration);
int offsetX = 0; int offsetY = 0;

toast.setGravity(Gravity.BOTTOM, offsetX, offsetY);
toast.show();
```

Если обычного текстового сообщения недостаточно, можно указать собственное Представление или разметку, чтобы использовать более сложный или визуально насыщенный способ отображения. Применяя метод setView, принадлежащий объекту Toast, вы можете задать любое Представление (включая разметку) для отображения с помощью механизма кратковременных уведомлений.

Например, в листинге 9.16 показана разметка, содержащая элементы TextView и CompassView из главы 4. Она будет выведена на экран в виде Toast.

Листинг 9.16. Использование Представлений для изменения уведомлений типа Toast

```
Context context = getApplicationContext();
String msg = "Cheers!";
int duration = Toast.LENGTH_LONG;
Toast toast = Toast.makeText(context, msg, duration);
toast.setGravity(Gravity.TOP, 0, 0);

LinearLayout ll = new LinearLayout(context);
ll.setOrientation(LinearLayout.VERTICAL);

TextView myTextView = new TextView(context);
CompassView cv = new CompassView(context);

myTextView.setText(msg);

int lHeight = LinearLayout.LayoutParams.FILL_PARENT;
int lWidth =  LinearLayout.LayoutParams.WRAP_CONTENT;

ll.addView(cv, new LinearLayout.LayoutParams(lHeight, lWidth));
```

```
ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight, lWidth));

ll.setPadding(40, 50, 0, 50);

toast.setView(ll);
toast.show();
```

Итоговое уведомление Toast будет выглядеть, как на рис. 9.2.

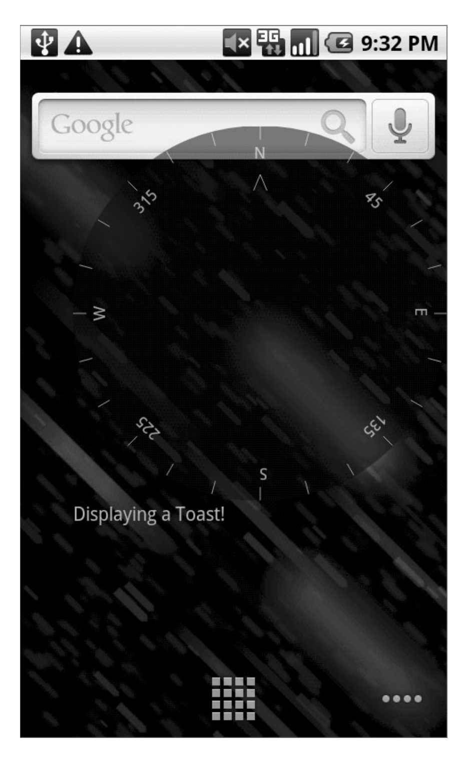


Рис. 9.2.

Использование уведомлений типа Toast в рабочих потоках

Как элемент графического интерфейса Toast должен быть вызван в потоке GUI, иначе существует риск выброса межпотокового исключения. В листинге 9.17 объект Handler используется для гарантии того, что уведомление Toast было вызвано в потоке GUI.

Листинг 9.17. Вызов Toast в потоке GUI

```
private void mainProcessing() {
  Thread thread = new Thread(null, doBackgroundThreadProcessing,
                             "Background");
```

Продолжение ⇨

Листинг 9.17 (продолжение)

```
    thread.start();
  }

  private Runnable doBackgroundThreadProcessing = new Runnable() {
    public void run() {
      backgroundThreadProcessing();
    }
  };

  private void backgroundThreadProcessing() {
    handler.post(doUpdateGUI);
  }

  // Объект Runnable, который вызывает метод из потока GUI
  private Runnable doUpdateGUI = new Runnable() {
    public void run() {
      Context context = getApplicationContext();
      String msg = "To open mobile development!";
      int duration = Toast.LENGTH_SHORT;
      Toast.makeText(context, msg, duration).show();
    }
  };
```

Знакомство с уведомлениями

Ваши приложения могут использовать уведомления, чтобы передавать пользователю какую-то информацию, не прибегая к помощи Активности. Уведомления управляются объектом NotificationManager и в настоящее время могут:

- создавать новые значки в строке состояния;
- отображать дополнительную информацию (и запускать Намерение) в расширенной строке для уведомлений;
- мерцать подсветкой или светодиодами;
- инициировать вибрацию;
- использовать звуковые оповещения (рингтоны, звуковые файлы из MediaStore).

Применение уведомлений — предпочтительный способ для невидимых компонентов приложения (Приемников широковещательных намерений, Сервисов и бездействующих Активностей) оповестить пользователя о событии, которому нужно уделить внимание. Уведомления также используются для отображения работы фоновых Сервисов, в частности тех, которые получили повышенный приоритет.

Как одна из абстракций в парадигме пользовательского интерфейса уведомления отлично подходят для мобильных устройств. Вполне вероятно,

что ваши пользователи не расстаются со своими телефонами, но вряд ли они постоянно уделяют им (или вашему приложению) внимание. В целом, у пользователей есть несколько приложений, работающих в фоновом режиме, и они не хотят следить за каждым из них.

Учитывая все вышесказанное, важно сделать так, чтобы ваши приложения могли оповещать пользователей об определенных событиях, которые требуют внимания.

Уведомления могут сохраняться путем регулярного повторения, получения статуса постоянных или с помощью отображения значка в статусной строке. Такие значки могут изменяться или расширяться для вывода дополнительной информации с помощью расширенной статусной строки, как показано на рис. 9.3.

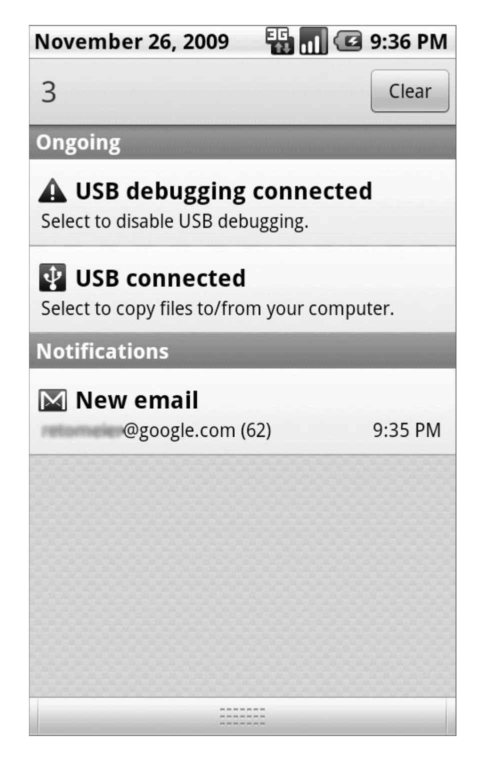


Рис. 9.3.

ПРИМЕЧАНИЕ

Чтобы отобразить расширенную статусную строку, щелкните на значке, который там находится, и потяните вниз экрана. Для того чтобы сохранить ее в таком состоянии, убедитесь, что вы отпустили ее уже после того, как она заняла весь экран. Если нужно спрятать расширенную статусную строку, потяните вверх.

Знакомство с NotificationManager

Класс NotificationManager — это системный `Сервис`, созданный для управления уведомлениями. Чтобы получить ссылку на него, вызовите метод getSystemService, как показано в листинге 9.18.

Листинг 9.18. Использование NotificationManager

```
String svcName = Context.NOTIFICATION_SERVICE;

NotificationManager notificationManager;
notificationManager = (NotificationManager)getSystemService(svcName);
```

Используя NotificationManager, вы можете создавать новые уведомления, изменять уже существующие или удалять те, в которых больше нет нужды.

Создание уведомлений

Android предоставляет несколько способов доносить информацию до пользователя с помощью уведомлений:

- значок в статусной строке;
- уведомление в расширенной статусной строке;
- дополнительные телефонные функции, такие как звонок или вибрация.

В этом разделе вы изучите два первых способа, далее в этой главе вы узнаете, как улучшить уведомления с помощью различных свойств объекта Notification, чтобы активизировать светодиоды устройства, виброзвонок или проигрывание звукового файла.

Создание уведомления и настройка значка в статусной строке

Начните с создания нового объекта Notification, передав ему в качестве параметров значок, который будет отображаться в статусной строке, текст сообщения и время, когда уведомление появится на экране, как показано в листинге 9.19.

Листинг 9.19. Создание уведомления

```
// Выберите графический объект, который будет отображаться в качестве
// значка в статусной строке
int icon = R.drawable.icon;
// Текст, который будет виден в статусной строке в момент появления
// уведомления
String tickerText = "Notification";
// В расширенной статусной строке уведомления сортируются по времени
// появления
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText, when);
```

Текст из переменной tickerText будет прокручиваться в статусной строке в момент появления уведомления.

Вы также можете задать для объекта Notification свойство number, чтобы отобразить количество событий на значке в статусной строке. Если это значение больше 1, как показано в следующем фрагменте кода, поверх значка выведется небольшое число:

```
notification.number++;
```

Чтобы изменение вступило в силу, нужно повторно вызвать объект Notification (это касается любых изменений); чтобы убрать число, выводимое поверх значка, установите значение в 0 или –1.

Настройка отображения уведомлений в расширенной статусной строке

Вы можете настраивать внешний вид уведомлений в расширенной статусной строке двумя способами:

- использовать метод setLatestEventInfo для обновления информации, которая отображается в расширенной статусной строке;
- установить новые значения для свойств contentView и contentIntent, чтобы уведомление отображалось нестандартным образом с помощью класса RemoteViews.

Проще всего наполнять данными разметку по умолчанию, применяя метод setLatestEventInfo. Обычная разметка, использующаяся в расширенной статусной строке, представляет собой значок, время, заголовок и описание. Все это передается в конструктор уведомления и метод setLatestEventInfo, как показано на рис. 9.4.

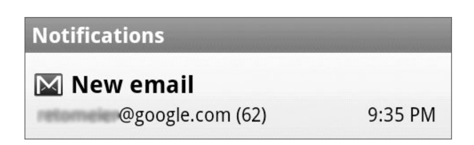


Рис. 9.4.

Уведомления часто дают возможность выполнять определенные действия, поэтому вы можете передать в метод setLatestEventInfo Намерение PendingIntent, которое будет срабатывать при нажатии на уведомление. Как правило, это Намерение должно открывать ваше приложение и направлять пользователя к Активности, которая предоставляет контекст для уведомления (например, показ непрочитанного SMS или электронного письма).

В листинге 9.20 продемонстрировано, каким образом передавать значения методу setLatestEventInfo.

Листинг 9.20. Установка значений для уведомления

```
Context context = getApplicationContext();
// Текст, который будет отображаться в расширенной статусной строке
String expandedText = "Extended status text";
// Заголовок для уведомления в расширенной статусной строке
String expandedTitle = "Notification Title";
// Намерение, загружающее Активность при нажатии на уведомление
Intent intent = new Intent(this, MyActivity.class);
PendingIntent launchIntent = PendingIntent.getActivity(context, 0,
intent, 0);

notification.setLatestEventInfo(context,
                                expandedTitle,
                                expandedText,
                                launchIntent);
```

Рекомендуется использовать один значок уведомления для представления нескольких экземпляров одного и того же события (например, получение нескольких SMS). Чтобы реализовать это, обновите значения, переданные в метод setLatestEventInfo для отображения последнего сообщения (или сводки нескольких сообщений), и повторно запустите уведомление. После этого обновятся все отображаемые значения.

Если степень детализации, которую предоставляет стандартная расширенная статусная строка, не подходит для вашего уведомления (или ее недостаточно), вы можете создать собственную разметку и назначить ее для своего уведомления, используя класс RemoteViews. На рис. 9.5 показана нестандартная разметка, которая описывается, назначается и изменяется в листингах 9.21, 9.22 и 9.23 соответственно.

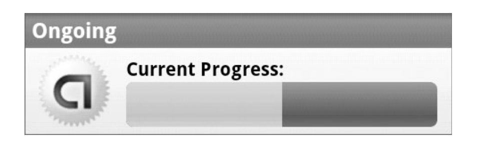


Рис. 9.5.

В листинге 9.21 описывается нестандартная разметка, содержащая значок, TextView и индикатор выполнения операций.

Листинг 9.21. Создание нестандартной разметки для уведомления в статусной строке

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:padding="5dp"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
```

```
  <ImageView
    android:id="@+id/status_icon"
    android:layout_width="wrap_content" android:layout_height="fill_
parent"
    android:layout_alignParentLeft="true"
  />
  <RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="10px"
    android:layout_toRightOf="@id/status_icon">
    <TextView
      android:id="@+id/status_text"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" android:layout_
alignParentTop="true"
      android:textColor="#000"
      android:textSize="14sp"
      android:textStyle="bold"
    />
    <ProgressBar
      android:id="@+id/status_progress"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_below="@id/status_text"
      android:progressDrawable="@android:drawable/progress_horizontal"
      android:indeterminate="false"
      android:indeterminateOnly="false"
    />
  </RelativeLayout>
</RelativeLayout>
```

Чтобы назначить данную разметку для уведомления, создайте новый объект RemoteViews и присвойте его свойству contentView. Необходимо также задать Ожидающее намерение свойству contentIntent. В листинге 9.22 показано, как назначить нестандартное Представление для только что созданного уведомления.

Листинг 9.22. Назначение нестандартной разметки для уведомления в статусной строке

```
Notification notification = new Notification(R.drawable.icon,
                                             "Custom Content",
                                             System.currentTimeMillis());
notification.flags = notification.flags | Notification.FLAG_ONGOING_EVENT;

notification.contentView = new RemoteViews(this.getPackageName(),
  R.layout.my_status_window_layout);

Intent intent = new Intent(this, MyActivity.class);
PendingIntent.getActivity(this, 0, intent, 0));
notification.contentIntent = pendingIntent;
```

ВНИМАНИЕ

Устанавливая значения для свойства contentView вручную, вы также должны присвоить значение для свойства contentIntent, иначе при показе уведомления будет выброшено исключение.

RemoteViews — механизм, позволяющий встраивать и управлять встроенными в отдельное приложение разметками. Чаще всего эти возможности применяются при создании виджетов для домашнего экрана. Набор Представлений, которые вы можете использовать при создании разметки, служащей параметром для конструктора RemoteViews, жестко ограничен. В следующей главе эта тема раскрыта более подробно.

Чтобы изменить свойства и внешний вид Представлений, которые стали частью разметки, описывающей внешний вид уведомления для статусной строки, используйте методы set*, принадлежащие объекту RemoteViews. В листинге 9.23 показывается, как менять свойства каждого Представления из разметки, которую вы определили в листинге 9.21.

Листинг 9.23. Изменение разметки уведомления для расширенной статусной строки

```
notification.contentView.setImageViewResource(R.id.status_icon,
                                              R.drawable.icon);
notification.contentView.setTextViewText(R.id.status_text,
                                         "Current Progress:");
notification.contentView.setProgressBar(R.id.status_progress,
                                        100, 50, false);
```

Этот метод особенно полезен в случае с текущими событиями (такими как процесс загрузки или воспроизведение файлов мультимедиа), когда необходимо передавать информацию о состоянии выполнения задачи. Далее в этой главе вы узнаете больше о текущих уведомлениях.

Показ уведомлений

Чтобы вывести уведомление на экран, передайте его в качестве параметра для метода notify (наряду с целочисленным ссылочным идентификатором), принадлежащего объекту NotificationManager, как показано в листинге 9.24.

Листинг 9.24. Показ уведомления

```
int notificationRef = 1;
notificationManager.notify(notificationRef, notification);
```

Чтобы обновить уведомление, которое уже было выведено на экран, вызовите его повторно с помощью NotificationManager, передавая методу notify тот самый ссылочный идентификатор. Вы даже можете передать тот же или совершенно новый объект уведомления. Пока не меняется значение

идентификатора, новое уведомление будет использовано для замены значка и уведомления, помещенного в расширенную статусную строку.

Вы также можете использовать ссылочный идентификатор для отмены показа уведомления. Для этого вызовите метод cancel, принадлежащий объекту NotificationManager:

```
notificationManager.cancel(notificationRef);
```

При отмене уведомление удаляется из расширенной статусной строки вместе со значком.

Внедрение уведомлений типа Notification и Toast в приложение Earthquake

В следующем примере[1] вы улучшите Сервис EarthquakeService, добавив в него показ уведомлений о новых землетрясениях. Кроме отображения значка в статусной строке уведомление будет показывать магнитуду и местоположение последнего землетрясения, а при нажатии на него станет открываться Активность Earthquake.

1. Начните с создания поля класса Notification внутри EarthquakeService, оно будет хранить объект для управления значком и уведомлением в расширенной статусной строке.

```
private Notification newEarthquakeNotification;
public static final int NOTIFICATION_ID = 1;
```

2. Расширьте метод onCreate, добавив создание объекта Notification.

```
@Override
public void onCreate() {
  updateTimer = new Timer("earthquakeUpdates");

  int icon = R.drawable.icon;
  String tickerText = "New Earthquake Detected";
  long when = System.currentTimeMillis();

  newEarthquakeNotification= new Notification(icon,
                                              tickerText,
                                              when);
}
```

3. Вернитесь к реализации EarthquakeLookupTask. Отредактируйте заглушку onProgressUpdate, чтобы уведомления вызывались при каждом добавлении нового землетрясения в Источник данных. Перед тем как вызвать уведомление, обновите расширенную информацию

[1] Все фрагменты кода в этом примере — часть проекта Earthquake 3 из главы 9, их можно загрузить с сайта Wrox.com.

с помощью метода setLatestEventInfo, создайте и отобразите новое уведомление типа Toast, чтобы сообщить о новом землетрясении.

```
@Override
protected void onProgressUpdate(Quake... values) {
  String svcName = Context.NOTIFICATION_SERVICE;
  NotificationManager notificationManager =
    (NotificationManager)getSystemService(svcName);

  Context context = getApplicationContext();
  String expandedText = values[0].getDate().toString();
  String expandedTitle = "M:" + values[0].getMagnitude() + " " +
                         values[0].getDetails();
  Intent startActivityIntent = new Intent(EarthquakeService.this,
                                          Earthquake.class);
  PendingIntent launchIntent =
    PendingIntent.getActivity(context, 0, startActivityIntent, 0);

  newEarthquakeNotification.setLatestEventInfo(context,
                                               expandedTitle,
                                               expandedText,
                                               launchIntent);
  newEarthquakeNotification.when =
    java.lang.System.currentTimeMillis();

  notificationManager.notify(NOTIFICATION_ID,
                             newEarthquakeNotification);

  Toast.makeText(context, expandedTitle, Toast.LENGTH_SHORT).show();
}
```

4. В завершение нужно очистить и отключить уведомления внутри двух классов Активностей. Это делается, чтобы значок уведомления не отображался, когда приложение активно.

   4.1. Начните с Активности Earthquake. Отредактируйте метод onCreate, чтобы получить ссылку на объект NotificationManager:

```
NotificationManager notificationManager;

@Override
public void onCreate(Bundle savedInstanceState) {
  [ ... ранее написанный код ... ]

  String svcName = Context.NOTIFICATION_SERVICE;
  notificationManager =
    (NotificationManager)getSystemService(svcName);
}
```

   4.2. Измените метод onReceive из EarthquakeReceiver. Поскольку объект этого класса регистрируется только в том случае, когда Активность находится на переднем плане, в этом участке кода вы можете безопасно отменять все уведомления о землетрясениях, как только они вызываются:

```
@Override
public void onReceive(Context context, Intent intent) {
  loadQuakesFromProvider();

  notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);
}
```

4.3. Далее дополните метод onResume, чтобы отменять уведомления в момент, когда Активность выходит на передний план:

```
@Override
public void onResume() {
  notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);

  IntentFilter filter;
  filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);

  receiver = new EarthquakeReceiver();
  registerReceiver(receiver, filter);
  super.onResume();
}
```

4.4. Проделайте то же самое с Активностью EarthquakeMap:

```
NotificationManager notificationManager;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.earthquake_map);

  ContentResolver cr = getContentResolver();
  earthquakeCursor = cr.query(EarthquakeProvider.CONTENT_URI,
                              null, null, null, null);

  MapView earthquakeMap = (MapView)findViewById(R.id.map_view);
  earthquakeMap.getOverlays().add(new
    EarthquakeOverlay(earthquakeCursor));

  String svcName = Context.NOTIFICATION_SERVICE;
  notificationManager = (NotificationManager)getSystemService(svcName);
}

@Override
public void onResume() {
  notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);

  earthquakeCursor.requery();

  IntentFilter filter;
  filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
  receiver = new EarthquakeReceiver();
  registerReceiver(receiver, filter);

  super.onResume();
```

```
}

public class EarthquakeReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);

    earthquakeCursor.requery();
    MapView earthquakeMap = (MapView)findViewById(R.id.map_view);
    earthquakeMap.invalidate();
  }
}
```

Продвинутые способы работы с уведомлениями

В следующих разделах вы узнаете, как улучшить уведомления, чтобы обеспечить дополнительные возможности по оповещению пользователя с помощью аппаратных средств, в частности звонка, светодиодов и вибрации.

После того как вы рассмотрите все эти улучшения, получите фрагмент кода, который может быть добавлен в проект Earthquake, чтобы позволить пользователю отсылать данные о силе обнаруженного землетрясения.

ПРИМЕЧАНИЕ

Для использования описанных в разделе методов отмените уведомление сразу после вызова, не показывая при этом значок в статусной строке, что позволит не выводить значок на экран, но никак не повлияет на другие действия.

Использование настроек по умолчанию

Самый простой и логичный способ добавить вибрацию, звуковой сигнал или мерцание светодиодами для ваших уведомлений — использование настроек по умолчанию. В свойстве defaults вы можете сочетать константы:

- Notification.DEFAULT_LIGHTS;
- Notification.DEFAULT_SOUND;
- Notification.DEFAULT_VIBRATE.

В следующем фрагменте кода к уведомлению добавляются звук и вибрации по умолчанию:

```
notification.defaults = Notification.DEFAULT_SOUND |
                        Notification.DEFAULT_VIBRATE;
```

Если хотите установить все значения по умолчанию, задействуйте константу Notification.DEFAULT_ALL.

Звуковое сопровождение

Использование звуковых оповещений для уведомления пользователя о событиях, связанных с устройством (таких как входящий звонок), — метод, который появился задолго до мобильных телефонов и выдержал испытание временем. Большинство стандартных событий, от входящих звонков до новый сообщений и низкого заряда батареи, объявляются с помощью звуковых мелодий.

Android позволяет проигрывать любой звуковой файл на телефоне в качестве уведомления. Чтобы это сделать, нужно присвоить свойству sound путь URI, как показано в следующем фрагменте:

```
notification.sound = ringURI;
```

Для применения собственного звукового файла загрузите его на устройство или добавьте в проект в качестве ресурса, как описано в главе 11.

Следующий фрагмент кода может быть отправлен в метод announceNewQuake, принадлежащий Сервису EarthquakeService из ранее рассмотренного примера. Он добавляет звуковую составляющую в уведомления о землетрясениях, проигрывая телефонный рингтон по умолчанию, если приходит информация о крупном землетрясении (магнитуда которого больше 6):

```
if (quake.getMagnitude() > 6) {
  Uri ringURI =
    RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
  newEarthquakeNotification.sound = ringURI;
}
```

Телефонный виброзвонок

Вы можете использовать функцию виброзвонка в телефоне, чтобы сопровождать ваше уведомление вибрацией.

Android позволяет разработчику управлять виброзвонком, можно применить вибрацию и для передачи информации, и для привлечения внимания пользователя.

Чтобы использовать виброзвонок, передайте в свойство vibrate объекта Notification массив значений типа long. Постройте массив, учитывая, что значения, отвечающие за продолжительность вибрации (в миллисекундах), чередуются со значениями, которые означают длину паузы между вибрациями.

Прежде чем использовать виброзвонок в своем приложении, необходимо получить для этого полномочия. Добавьте тег `uses-permission` в манифест приложения, чтобы запросить доступ к виброзвонку устройства:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

В следующем примере показано, как изменить уведомление, чтобы одна секунда вибрации сменялась одной секундой паузы на протяжении пяти секунд:

```
long[] vibrate = new long[] { 1000, 1000, 1000, 1000, 1000 };
notification.vibrate = vibrate;
```

Вы можете воспользоваться этой тонкой возможностью для передачи пользователям информации. В следующем дополнении к методу announceNewQuake телефон будет вибрировать, учитывая силу землетрясения. Землетрясения измеряются по экспоненциальной шкале, поэтому примените тот же метод для создания модели поведения виброзвонка.

В случае с землетрясениями, имеющими магнитуду 1, телефон будет вибрировать долю секунды; при землетрясении с магнитудой 10, когда земля расколется пополам, ваши пользователи будут иметь преимущество перед началом Апокалипсиса, так как их телефоны станут вибрировать целых 20 секунд. Наиболее распространены землетрясения 3–7 баллов по шкале Рихтера, что «выльется» в виброзвонок длиной от 200 миллисекунд до 4 секунд:

```
double vibrateLength = 100*Math.exp(0.53*quake.getMagnitude());
long[] vibrate = new long[] {100, 100, (long)vibrateLength };
newEarthquakeNotification.vibrate = vibrate;
```

ПРИМЕЧАНИЕ

В настоящее время эмулятор Android не умеет оповещать о вибрации ни визуально, ни с помощью звуковых сигналов.

Индикация с помощью светодиодов

Объект Notification включает в себя свойства для настройки цвета и частоты мерцания светодиодов устройства.

ВНИМАНИЕ

У каждого устройства могут быть свои ограничения в контроле над светодиодами. Если не удастся указать цвет, используются наиболее приближенные возможности. Не забывайте об этих ограничениях, когда задействуете светодиоды для передачи информации пользователю, и не делайте этот способ единственно доступным.

Свойство ledARGB может устанавливать цвет для светодиодной подсветки. Свойства ledOffMS и ledOnMS позволяют регулировать частоту и поведение светодиодов. Вы можете включить светодиоды, присвоив свойству ledOnMS значение 1, а ledOffMS – 0. Присвоив им обоим значения 0, светодиоды можно выключить.

Настроив работу со светодиодами, необходимо также добавить флаг FLAG_SHOW_LIGHTS к свойству flags объекта Notification.

В следующем фрагменте кода показано, как включить на устройстве красный светодиод:

```
notification.ledARGB = Color.RED;
notification.ledOffMS = 0;
notification.ledOnMS = 1;
notification.flags = notification.flags | Notification.FLAG_SHOW_LIGHTS;
```

Контроль над цветом и частотой мерцания дает вам одну возможность для передачи информации пользователям.

В случае с отслеживанием землетрясений вы можете использовать светодиоды, чтобы помочь пользователям воспринимать нюансы экспоненциальной шкалы при сообщении о магнитуде. В следующем фрагменте цвет светодиодов зависит от силы землетрясения, а частота мерцания обратно пропорциональна ей:

```
int color;
if (quake.getMagnitude() < 5.4)
  color = Color.GREEN;
else if (quake.getMagnitude() < 6)
  color = Color.YELLOW;
else
  color = Color.RED;

newEarthquakeNotification.ledARGB = color;
newEarthquakeNotification.ledOffMS = (int)vibrateLength;
newEarthquakeNotification.ledOnMS = (int)vibrateLength;
newEarthquakeNotification.flags = newEarthquakeNotification.flags |
                                  Notification.FLAG_SHOW_LIGHTS;
```

ПРИМЕЧАНИЕ

В настоящее время эмулятор Android не умеет визуально иллюстрировать активность светодиодов.

Текущие и настойчивые уведомления

Вы можете делать уведомления текущими и/или настойчивыми, устанавливая флаги FLAG_INSISTENT и FLAG_ONGOING_EVENT.

Уведомления, помеченные как текущие, используются для представления событий, которые выполняются в данный момент времени (например, загрузка файла, фоновое проигрывание музыки). Текущие уведомления необходимы для Сервисов, работающих на переднем плане, о которых шла речь ранее в этой главе. Пример установки флагов:

```
notification.flags = notification.flags |
  Notification.FLAG_ONGOING_EVENT;
```

В расширенной статусной строке текущие события отделены от обычных, как показано на рис. 9.6.

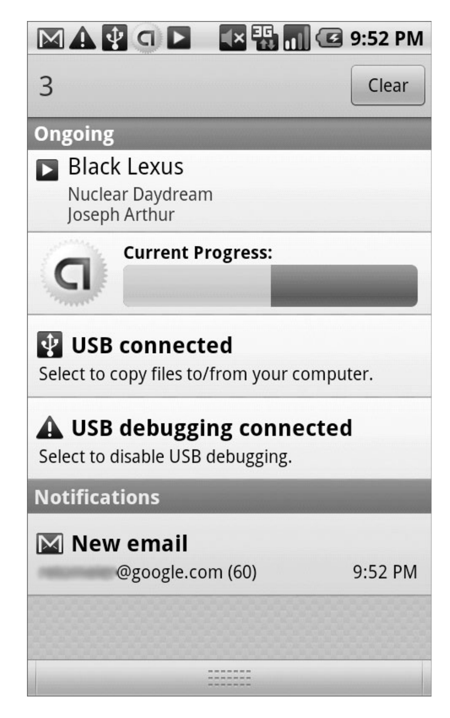


Рис. 9.6.

Настойчивые уведомления непрерывно повторяют звуковые сигналы, вибрируют и мерцают светодиодами, пока не будут остановлены. Подобные уведомления, как правило, используются для событий, которые требуют немедленного и своевременного внимания, таких как входящий звонок или срабатывание будильника.

В следующем фрагменте кода показано, как сделать уведомление настойчивым:

```
notification.flags = notification.flags |
  Notification.FLAG_INSISTENT;
```

Использование Сигнализации

Механизм Сигнализации не зависит от конкретного приложения, это значит, что Намерения срабатывают в определенное время или с заданным интервалом.

Сигнализация устанавливается за пределами вашего приложения, поэтому она может быть использована для вызова событий, принадлежащих

приложению, или для запуска самого приложения, даже если оно закрыто. Механизм Сигнализации может стать особенно мощным инструментом в сочетании с Приемниками широковещательных намерений. Вы имеете возможность устанавливать Сигнализацию, которая будет вызывать Широковещательные намерения, запускать Сервисы или даже открывать Активности, даже если приложение закрыто.

Таким образом, Сигнализация чрезвычайно полезна, когда речь идет о снижении требований к ресурсам, особенно в случае с приложениями, которые работают в фоновом режиме. Она позволяет останавливать Сервисы и отключать таймеры, сохраняя возможность выполнять запланированные действия.

Например, вы можете использовать Сигнализацию для создания приложения-будильника, выполнения регулярных сетевых запросов, запуска трудоемких или дорогих операций, запланированных на определенное время.

ПРИМЕЧАНИЕ

Для планирования операций, которые работают только на протяжении жизненного цикла вашего приложения, вместо механизма Сигнализации лучше использовать класс Handler в сочетании с таймерами и потоками. Это позволит Android лучше контролировать системные ресурсы. Сигнализация сокращает жизненный цикл вашего приложения, вынося запланированные события из-под его контроля.

Сигнализация в Android остается активной даже тогда, когда устройство находится в режиме ожидания и при необходимости может его «пробудить», однако она отменяется каждый раз, когда устройство перезагружается.

Задания, выполняющиеся Сигнализацией, управляются с помощью объекта AlarmManager, системного Сервиса, доступного через вызов метода getSystemService:

```
AlarmManager alarms =
  (AlarmManager)getSystemService(Context.ALARM_SERVICE);
```

Чтобы создать Сигнализацию, которая сработает всего один раз, используйте метод set, укажите тип Сигнализации, время срабатывания и Ожидающее намерение, которое должно запуститься. Если время срабатывания, которое вы указали для Сигнализации, уже прошло, Намерение запустится немедленно.

Разработчику доступны четыре вида Сигнализации. В зависимости от вашего выбора значение, переданное в метод set, будет означать или конкретное время, или время задержки.

- RTC_WAKEUP. Выводит устройство из режима ожидания для запуска Ожидающего намерения в указанное время.

- RTC. Запускает Ожидающее намерение в указанное время, но не выводит устройство из режима ожидания.
- ELAPSED_REALTIME. Запускает Ожидающее намерение, основываясь на времени, которое прошло с момента загрузки устройства, но не с момента выхода из режима ожидания. Это время включает любой временной промежуток, в котором устройство находилось в данном режиме. Обратите внимание, что прошедшее время вычисляется на основании того, когда устройство было загружено.
- ELAPSED_REALTIME_WAKEUP. По прошествии указанного промежутка времени с момента загрузки выводит устройство из спящего режима и запускает Ожидающее намерение.

Процесс создания Сигнализации показан в листинге 9.25.

Листинг 9.25. Создание Сигнализации

```
int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;
long timeOrLengthofWait = 10000;
String ALARM_ACTION = "ALARM_ACTION";
Intent intentToFire = new Intent(ALARM_ACTION);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0,
  intentToFire, 0);

alarms.set(alarmType, timeOrLengthofWait, pendingIntent);
```

После срабатывания Сигнализации Ожидающее намерение, которое вы указали, начнет транслироваться. Установка еще одной Сигнализации с тем же Ожидающим намерением приведет к замене предыдущей Сигнализации.

Для того чтобы отменить Сигнализацию, вызовите метод cancel из объекта AlarmManager, передав ему Ожидающее намерение, которое больше не должно срабатывать, как показано в следующем фрагменте:

```
alarms.cancel(pendingIntent);
```

В листинге 9.26 устанавливаются два экземпляра Сигнализации, и первый в итоге отменяется. Для первого экземпляра время, когда он должен сработать, определяется явно. Второй же должен сработать через 30 минут после загрузки устройства, но он не выведет устройство из режима ожидания, если оно в нем находится.

Листинг 9.26. Установка и отмена Сигнализации

```
AlarmManager alarms =
(AlarmManager)getSystemService(Context.ALARM_SERVICE);

String MY_RTC_ALARM = "MY_RTC_ALARM";
String ALARM_ACTION = "MY_ELAPSED_ALARM";

PendingIntent rtcIntent =
```

```
    PendingIntent.getBroadcast(this, 0,
                               new Intent(MY_RTC_ALARM), 1);
  PendingIntent elapsedIntent =
    PendingIntent.getBroadcast(this, 0,
                               new Intent(ALARM_ACTION), 1);

  // Пробудить устройство и запустить Намерение через 5 часов.
  Date t = new Date();
  t.setTime(java.lang.System.currentTimeMillis() + 60*1000*5);
  alarms.set(AlarmManager.RTC_WAKEUP, t.getTime(), rtcIntent);

  // Запустить Намерение через 30 минут, если устройство не находится
  // в спящем режиме.
  alarms.set(AlarmManager.ELAPSED_REALTIME, 30*60*1000, elapsedIntent);

  // Отменить первую Сигнализацию.
  alarms.cancel(rtcIntent);
```

Установка повторяющейся Сигнализации

AlarmManager позволяет устанавливать повторяющуюся Сигнализацию, когда необходимо регулярно выполнять запланированные действия.

Повторяющаяся Сигнализация работает точно так же, как и одиночная, описанная ранее, но она продолжает срабатывать через определенные промежутки времени, пока ее не отменят.

Поскольку Сигнализация находится вне контекста вашего приложения, она идеально подходит для регулярных обновлений или поиска данных, поэтому ей не нужен Сервис, постоянно работающий в фоновом режиме.

Чтобы установить повторяющуюся Сигнализацию, используйте метод setRepeating или setInexactRepeating из объекта AlarmManager, как показано в листинге 9.27. Оба метода поддерживают тип Сигнализации, начальный момент срабатывания и Ожидающее намерение, которое должно запускаться (как описывалось в предыдущем разделе).

Применяйте метод setRepeating в тех случаях, когда нужно контролировать точный интервал между срабатываниями Сигнализации. Значение интервала, переданное в данный метод, дает возможность определить точные временные промежутки, вплоть до миллисекунды.

Метод setInexactRepeating предоставляет мощный инструмент для уменьшения энергозатрат, связанных с выходом устройства из режима ожидания для выполнения запланированных действий. Вместо точного интервала этот метод принимает одну из следующих констант, принадлежащих объекту AlarmManager:

- INTERVAL_FIFTEEN_MINUTES;
- INTERVAL_HALF_HOUR;
- INTERVAL_HOUR;

- INTERVAL_HALF_DAY;
- INTERVAL_DAY.

Во время работы Android синхронизирует некоторые неточно повторяющиеся экземпляры Сигнализации и запускает их одновременно. Это помогает избежать ситуации, когда каждое приложение отдельно пытается вывести устройство из режима ожидания. Система «пробуждает» устройство со сходными, но не пересекающимися интервалами, чтобы обновиться или запросить новые данные по Сети. Выполняя данную синхронизацию, она имеет возможность ограничить влияние регулярно происходящих событий на заряд батареи.

Листинг 9.27. Установка повторяющейся Сигнализации

```
// Запускать Намерение в точности каждый час, если устройство выведено
// из  режима ожидания.
alarms.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                   60*60*1000, 60*60*1000, elapsedIntent);

// Вывести устройство из спящего режима и запускать Сигнализацию
// каждый час
alarms.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                           60*60*1000, AlarmManager.INTERVAL_DAY,
                           elapsedIntent);
```

ВНИМАНИЕ

Воздействие регулярно повторяющихся событий на время работы батареи может быть весьма значительным. На практике старайтесь как можно больше ограничивать частоту срабатывания Сигнализации, выводите устройство из режима ожидания только в том случае, если это действительно необходимо, используйте неточно повторяющиеся экземпляры Сигнализации там, где это возможно.

Использование повторяющейся Сигнализации для обновления приложения Earthquake

В этом — последнем — дополнении к примеру Earthquake[1] вы замените Сигнализацей таймер, который в настоящее время используется для планирования сетевых обновлений данного приложения.

Одно из наиболее важных преимуществ такого подхода в том, что Сервис теперь сможет останавливаться после завершения обновления, освобождая ценные системные ресурсы.

1. Начните с создания нового класса EarthquakeAlarmReceiver, наследующего BroadcastReceiver.

[1] Все фрагменты кода в этом примере — часть проекта Earthquake 4 из главы 9, их можно загрузить с сайта Wrox.com.

```
package com.paad.earthquake;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class EarthquakeAlarmReceiver extends BroadcastReceiver {

}
```

2. Переопределите метод onReceive, для того чтобы явно запускать Сервис EarthquakeService.

```
@Override
public void onReceive(Context context, Intent intent) {
  Intent startIntent = new Intent(context, EarthquakeService.class);
  context.startService(startIntent);
}
```

3. Создайте новое публичное статическое строковое свойство, чтобы определить действие, которое будет использовано при срабатывании Приемника широковещательных намерений.

```
public static final String ACTION_REFRESH_EARTHQUAKE_ALARM =
  "com.paad.earthquake.ACTION_REFRESH_EARTHQUAKE_ALARM";
```

4. Включите только что созданный EarthquakeAlarmReceiver в манифест, добавив тег `<intent-filter>`, с помощью которого происходит отслеживание действия, заданного в предыдущем пункте.

```
<receiver android:name=".EarthquakeAlarmReceiver">
  <intent-filter>
    <action
android:name="com.paad.earthquake.ACTION_REFRESH_EARTHQUAKE_ALARM"
    />
  </intent-filter>
</receiver>
```

5. Отредактируйте метод onCreate внутри EarthquakeService, чтобы получить ссылку на AlarmManager и создать новый объект PendingIntent, который запустится при срабатывании Сигнализации. Вы также можете убрать инициализацию timerTask.

```
AlarmManager alarms;
PendingIntent alarmIntent;

@Override
public void onCreate() {
  int icon = R.drawable.icon;
  String tickerText = "New Earthquake Detected";
  long when = System.currentTimeMillis();

  newEarthquakeNotification =
```

```
    new Notification(icon, tickerText, when);

  alarms = (AlarmManager)getSystemService(Context.ALARM_SERVICE);

  String ALARM_ACTION;
  ALARM_ACTION =
  EarthquakeAlarmReceiver.ACTION_REFRESH_EARTHQUAKE_ALARM;
  Intent intentToFire = new Intent(ALARM_ACTION);
  alarmIntent =
    PendingIntent.getBroadcast(this, 0, intentToFire, 0);
}
```

6. Отредактируйте метод onStartCommand, чтобы установить повторяющуюся Сигнализацию вместо использования таймера для запланированных обновлений (если автоматические обновления включены в настройках). Определение нового Намерения с тем же действием приведет к автоматической отмене любой другой установленной ранее Сигнализации.

   Используйте этот метод, чтобы изменить возвращаемый результат. Вместо Service.START_STICKY верните Service.START_NOT_STICKY. Далее нужно сделать так, чтобы Сервис останавливался по завершении фонового обновления. Использование Сигнализации гарантирует, что следующее обновление произойдет в соответствии с заданной частотой, поэтому системе не нужно повторно запускать Сервис, если его работа внезапно прервалась.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  SharedPreferences prefs =
    getSharedPreferences(Preferences.USER_PREFERENCE,
    Activity.MODE_PRIVATE);

  int minMagIndex = prefs.getInt(Preferences.PREF_MIN_MAG, 0);
  if (minMagIndex < 0)
    minMagIndex = 0;

  int freqIndex = prefs.getInt(Preferences.PREF_UPDATE_FREQ, 0);
  if (freqIndex < 0)
    freqIndex = 0;

  boolean autoUpdate =
    prefs.getBoolean(Preferences.PREF_AUTO_UPDATE, false);

  Resources r = getResources();
  int[] minMagValues = r.getIntArray(R.array.magnitude);
  int[] freqValues = r.getIntArray(R.array.update_freq_values);

  minimumMagnitude = minMagValues[minMagIndex];
  int updateFreq = freqValues[freqIndex];

  if (autoUpdate) {
    int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;
```

```
    long timeToRefresh = SystemClock.elapsedRealtime() +
                         updateFreq*60*1000;
    alarms.setRepeating(alarmType, timeToRefresh,
                        updateFreq*60*1000, alarmIntent);
  }
  else
    alarms.cancel(alarmIntent);

  refreshEarthquakes();

  return Service.START_NOT_STICKY;
};
```

7. Наполните кодом заглушку onPostExecute внутри Earthquake-LookupTask, чтобы вызывать метод stopSelf после завершения фонового обновления. Поскольку данная асинхронная задача вызывается исключительно из обработчика onStartCommand и только если она еще не запущена, это гарантирует, что работа Сервиса никогда не завершится преждевременно.

```
@Override
protected void onPostExecute(Void result) {
  stopSelf();
}
```

8. Удалите поле updateTimer и экземпляр задачи doRefresh.

Резюме

Фоновые Сервисы — один из наиболее веских доводов в пользу разработки приложений для платформы Android, но их использование вызывает некоторые сложности. В этой главе вы узнали, как задействовать эти невидимые программные компоненты для выполнения различных задач, в то время как ваши приложения работают в фоновом режиме.

Вы познакомились с уведомлениями типа Toast, кратковременными всплывающими сообщениями, которые позволяют отображать информацию для пользователей, не отвлекая их и не прерывая их работу с программой.

Вы использовали NotificationManager для того, чтобы слать уведомления пользователям из Сервисов и Активностей, применяя заранее настроенное мерцание светодиодов, виброзвонок и звуковые файлы для передачи подробной информации о событии. Вы узнали, как и когда нужно создавать текущие уведомления и каким образом настраивать их разметку для отображения в расширенной статусной строке.

С помощью Сигнализации получили возможность создавать и отменять запланированные события, делать их регулярными, чтобы передавать Намерения или запускать Сервисы.

В этой главе также рассказывалось:

- как привязывать Сервисы к Активностям, чтобы использовать более детализированные и структурированные интерфейсы вместо дополнительных свойств в Намерении;
- гарантировать, что ваше приложение останется отзывчивым, путем перемещения медленных или трудоемких задач (таких как сетевые запросы) в другой поток с помощью AsyncTask;
- использовать обработчики для синхронизации дочерних потоков с главным потоком GUI при выполнении различных действий, применяя визуальные элементы управления и уведомления типа Toast.

В главе 10 вы узнаете, как интегрировать ваше приложение в домашний экран Android, начиная с разработки динамических, интерактивных виджетов для домашнего экрана и заканчивая созданием Живых каталогов и Живых обоев для Рабочего стола. В завершение познакомитесь с Виджетом быстрого поиска и узнаете, как выводить в него поисковые результаты из своего приложения.

Глава 10
ДОМАШНИЙ ЭКРАН В ANDROID

Содержание главы

- Создание виджетов для домашнего экрана.
- Реализация Живых каталогов.
- Добавление поисковых возможностей в ваши приложения.
- Доставка поисковых результатов в Виджет быстрого поиска (Quick Search Box).
- Создание Живых обоев для домашнего экрана.

Виджеты, Живые каталоги, Живые обои и Виджет быстрого поиска позволяют получить собственную часть домашнего экрана пользователя, предоставляя либо удобную точку входа в приложение, либо автономный источник информации. Это потрясающее нововведение как для пользователей, так и для разработчиков. Оно позволяет:

- пользователям — получать постоянный доступ к интересующей их информации без необходимости запускать приложение;
- разработчикам — получать точку входа в свои приложения прямо на домашнем экране.

Полезный виджет Живой каталог или динамические обои для домашнего экрана снижают вероятность того, что приложение будет удалено, и делают его шансы на успех более высокими.

Получая новые возможности, не стоит забывать об ответственности. Виджеты на домашнем экране работают постоянно в виде дочерних процессов. Вы должны быть чрезвычайно осторожными при создании виджетов, следя за тем, чтобы они оставались отзывчивыми и не потребляли слишком много системных ресурсов.

В этой главе рассказывается, как создавать и использовать виджеты, Живые каталоги и Живые обои. Мы остановимся подробно на том, что они из себя представляют, как их использовать, каким образом привнести интерактивность в эти компоненты приложения.

Вы также узнаете, как интегрировать поисковый фреймворк Android в свое приложение и как доставлять поисковые результаты в Виджет быстрого поиска.

Знакомство с виджетами на домашнем экране

Виджеты (или AppWidgets, если быть более точным) — это визуальные компоненты приложения, которые могут быть добавлены в другие приложения. Наиболее яркий пример этого механизма — стандартный домашний экран в Android, на который пользователи могут добавлять виджеты, хотя при желании можно добавить их в любое свое приложение.

Виджеты дают возможность вашему приложению получить собственный кусочек домашнего экрана пользователя. Хороший виджет должен предоставлять полезную, лаконичную и своевременную информацию, используя при этом минимальное количество ресурсов.

Виджеты могут быть как автономными приложениями (стандартный виджет с часами), так и компактными, но довольно заметными компонентами родительских приложений (виджеты календаря или медиапроигрывателя).

На рис. 10.1 показано четыре стандартных виджета, размещенных на домашнем экране, которые доступны на устройствах под управлением Android: поиск, управление энергопотреблением, прогноз погоды и новости, а также плеер.

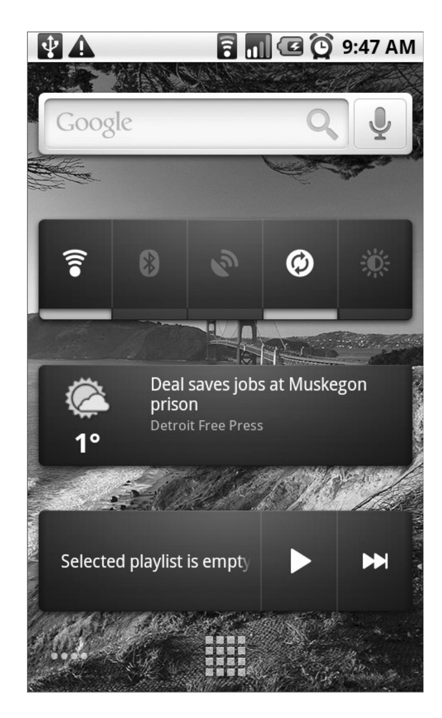


Рис. 10.1.

ПРИМЕЧАНИЕ

Чтобы добавить виджет на домашний экран, продолжительно нажимайте на свободном участке экрана, после чего выберите нужные компоненты. Откроется список доступных виджетов. После добавления можете перемещать их по экрану с помощью длительного нажатия и перетаскивания. Чтобы убрать виджеты, перетащите их на значок корзины внизу экрана.

Виджеты, которые были активизированы, управляются внутри процесса домашнего экрана. Они будут выводить устройство из режима ожидания в зависимости от частоты своих обновлений, чтобы всегда предоставлять актуальную информацию при попадании в зону видимости. Как разработчику вам необходимо проявлять особую осторожность при создании виджетов, обеспечивая минимальную частоту обновлений и их легковесность.

В следующих разделах говорится о том, как создавать виджеты, и приводятся некоторые рекомендации по обновлению и добавлению интерактивности.

Создание виджетов

Виджеты реализуются в виде Приемников широковещательных намерений. Они используют объекты RemoteViews для обновления иерархии Представлений, хранящейся внутри процесса другого приложения. Как правило, в качестве этого процесса выступает домашний экран.

Чтобы разработать виджет для своего приложения, сперва необходимо создать три компонента:

- ресурс разметки, который описывает пользовательский интерфейс виджета;
- файл в формате XML, описывающий метаданные, связанные с виджетом;
- объект IntentReceiver, который определяет и контролирует поведение виджета.

Число виджетов, которые вы можете создать для одного приложения, не ограничено. Все виджеты могут иметь как собственные, так и общие размер, разметку, частоту и логику обновления. Иногда полезно предоставлять несколько версий виджета с разными размерами.

Создание разметки для виджета

Первый шаг при создании вашего виджета — разработка и реализация его пользовательского интерфейса.

Процесс создания пользовательских интерфейсов для виджетов такой же, как и для других визуальных компонентов в Android, рассмотренных в главе 4. Лучше всего описать разметку вашего виджета во внешнем ресурсе с помощью XML, но это можно сделать и программным способом внутри метода onCreate, который принадлежит Приемнику намерений.

Руководство по проектированию виджетов

Поскольку на домашнем экране наряду со стандартными виджетами часто отображаются и сторонние, соблюдение стандартов проектирования чрезвычайно важно.

Разработаны принципы создания пользовательского интерфейса для виджетов, которые охватывают как размеры разметки, так и визуальную стилизацию. Размеры должны жестко контролироваться, тогда как стилистические вопросы можно назвать лишь рекомендациями. Все это рассматривается в следующих разделах. Более подробную информацию можное найти на странице «Руководство по проектированию виджетов» для разработчиков под Android по адресу http://developer.android.com/guide/practices/ui_guidelines/widget_design.html.

Разметка виджетов и их размеры. Домашний экран в Android по умолчанию разделен сеткой 4 × 4, каждая ячейка которой имеет размер 74 × 74 аппаратно независимых пикселов (dp). Чтобы выбрать ширину и высоту виджета, сперва определитесь с количеством ячеек, которые будете использовать. Общее число пикселов равно числу занятых ячеек, умноженному на 74, минус два пиксела на отступ от края рамки:

```
Минимальный размер в dp = (Количество ячеек * 74dp) - 2dp
```

Если это значение не соответствует точным размерам ячеек на домашнем экране, размеры вашего виджета округляются, чтобы полностью их заполнить.

Размеры виджета указываются в файле настроек, который будет описан в следующих разделах этой главы.

Визуальная стилизация виджета, который представляет ваше приложение на домашнем экране, — весьма важный аспект проектирования. Вы должны убедиться, что его стиль соответствует внешнему виду вашего приложения и других компонентов домашнего экрана.

Подробное описание стиля, который продвигает компания Google для виджетов, выходит за рамки данной книги, но обратите внимание на общие рекомендации для пользовательского интерфейса виджетов, ссылка на которые дана выше. В документации от Google описываются ресурсы с изображениями, задействованными при создании стандартных виджетов

для Android, которые поставляются с устройствами, лицензированными компанией Google.

Виджеты полностью поддерживают прозрачный фон и позволяют использовать изображения в форматах PNG и NinePatch.

Виды разметки и Представлений, которые можно использовать в виджетах

Из соображений безопасности и производительности установлены некоторые ограничения при выборе видов разметки и Представлений для создания пользовательского интерфейса виджета.

В целом следующие представления не могут быть использованы в разметке для виджетов (если вы все же попробуете их задействовать, то получите исключение NullPointerException):

- любые нестандартные Представления;
- потомки разрешенных Представлений;
- EditText.

В настоящее время для виджетов доступны только следующие виды разметки:

- FrameLayout;
- LinearLayout;
- RelativeLayout.

Представления, которые в них можно хранить:

- AnalogClock;
- Button;
- Chronometer;
- ImageButton;
- ImageView;
- ProgressBar;
- TextView.

Особенно полезными могут быть элементы TextView и ImageView. Позже в этой главе вы узнаете, как использовать ImageView в сочетании с ресурсом SelectionStateDrawable, чтобы создавать интерактивные виджеты с минимумом программного кода или вовсе без него.

В листинге 10.1 показан пример разметки, которая описывает пользовательский интерфейс виджета.

Листинг 10.1. Ресурс разметки для виджета в формате XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10sp">
  <ImageView
    android:id="@+id/widget_image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/icon"
  />
  <TextView
    android:id="@+id/widget_text"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="Text Goes Here"
  />
</LinearLayout>
```

Определение настроек для виджета

Описание ресурсов виджета хранится в формате XML внутри каталога res/xml вашего проекта. Тег `appwidget-provider` позволяет описывать метаданные виджета, которые определяют размер, разметку и частоту обновления с помощью нескольких атрибутов.

- `initialLayout`. Ресурс разметки, используемый для создания пользовательского интерфейса виджета.
- `minWidth` / `minHeight`. Минимальная ширина и минимальная высота виджета соответственно (описывались в предыдущем разделе).
- `label`. Название, которое будет отображаться в меню выбора виджетов.
- `updatePeriodMillis`. Минимальный период между обновлениями виджета (в миллисекундах). Android будет выводить устройство из режима ожидания для обновления вашего виджета с этой частотой, поэтому вы должны указать значение, равное как минимум одному часу. В идеале ваш виджет не должен использовать этот способ обновления чаще, чем раз или два в день. Этот и другие способы обновления более подробно рассмотрены ниже в данной главе.
- `configure`. При желании можете указать полноценную Активность, которая загрузится в момент добавления вашего виджета на домашний экран.

В листинге 10.2 показан файл ресурсов, который описывает виджет размером 2 × 2 ячейки, обновляющийся раз в час и использующий разметку, созданную в предыдущем разделе.

Листинг 10.2. Определение файла ресурсов для виджета

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:initialLayout="@layout/my_widget_layout"
  android:minWidth="146dp"
  android:minHeight="146dp"
  android:label="My App Widget"
  android:updatePeriodMillis="3600000"
/>
```

Создание Приемника намерений для вашего виджета и добавление его в манифест приложения

Виджет представляет собой реализацию Приемника широковещательных намерений и включает в себя Фильтры намерений, которые отслеживают объекты Intent. А они в свою очередь инициируют обновление виджета с помощью действий AppWidget.ACTION_APPWIDGET_UPDATE, DELETED, ENABLED и DISABLED. Вы можете создать свой виджет, напрямую наследовав класс IntentReceiver и переопределив метод onReceive для обработки поступающих Намерений.

Класс AppWidgetProvider упрощает данную задачу, скрывая от разработчика все операции с Намерениями и предоставляя обработчики для обновления, удаления, включения и отключения.

В листинге 10.3 показана реализация простого виджета, который наследует класс AppWidgetProvider и переопределяет метод onUpdate.

Листинг 10.3. Реализация виджета

```
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.Context;

public class MyAppWidget extends AppWidgetProvider {
  @Override
  public void onUpdate(Context context,
                       AppWidgetManager appWidgetManager,
                       int[] appWidgetIds) {
    // TODO Обновить пользовательский интерфейс виджета.
  }
}
```

В манифест приложения виджеты добавляются так же, как и другие Приемники намерений. Однако чтобы указать Приемник намерений в качестве виджета, вам необходимо вставить внутрь него два дополнительных тега (листинг 10.4):

- Фильтр намерений для действия android.appwidget.action.APPWIDGET_UPDATE;
- ссылку на ресурс с метаданными в формате XML, в котором описан ваш виджет.

Листинг 10.4. Описание виджета в манифесте приложения

```
<receiver android:name=".MyAppWidget" android:label="My App Widget">
  <intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>
  <meta-data
    android:name="android.appwidget.provider"
    android:resource="@xml/my_app_widget_info"
  />
</receiver>
```

Знакомство с RemoteViews и AppWidgetManager

Класс RemoteViews предназначен для описания и управления иерархиями Представлений, которые принадлежат к процессу другого приложения. Это позволяет изменять свойства или вызывать методы, принадлежащие Представлению, которое выступает частью другого приложения.

Например, Представления внутри виджетов работают в отдельном процессе (как правило, это домашний экран), поэтому RemoteViews может использоваться для изменения пользовательского интерфейса виджета из Приемника намерений, работающего внутри вашего приложения.

Класс AppWidgetManager задействуется для обновления виджетов и предоставляет информацию о них.

Используя RemoteViews в сочетании с AppWidgetManager, вы получите возможность изменять внешний вид Представлений, которые поддерживаются фреймворком для создания виджетов. Среди прочего можно показывать и скрывать элементы, менять текст и изображения, добавлять реакцию на события нажатия.

В этом разделе рассказывается о том, как создавать новые объекты RemoteViews внутри и за пределами метода onUpdate, принадлежащего классу AppWidgetProvider. Вы также узнаете, как использовать RemoteViews для обновления пользовательского интерфейса виджетов и добавления к ним интерактивности.

Создание объектов RemoteViews и использование AppWidgetManager для их применения

Чтобы создать новый объект класса RemoteViews, необходимо передать в его конструктор имя пакета приложения, откуда он вызывается, и ресурс разметки, с которой вы хотите работать, как показано в листинге 10.5. Позже в этом разделе вы научитесь использовать объект RemoteViews для обновления Представлений и разметки вашего виджета.

Листинг 10.5. Использование RemoteViews

```
RemoteViews views = new RemoteViews(context.getPackageName(),
                                    R.layout.my_remote_layout);
```

Чтобы манипулировать виджетами с помощью RemoteViews, вызовите статический метод getInstance. В итоге вы получите экземпляр AppWidgetManager, который можно использовать для поиска каждого экземпляра конкретного виджета. В продолжении листинга 10.5 показано, как это делать:

```
// Получите экземпляр AppWidgetManager.
AppWidgetManager appWidgetManager = AppWidgetManager.
getInstance(context);
// Получите идентификаторы каждого экземпляра выбранного виджета.
ComponentName thisWidget = new ComponentName(context, MyAppWidget.class);
int[] appWidgetIds = appWidgetManager.getAppWidgetIds(thisWidget);
```

После того как вы закончите вносить изменения в объект RemoteViews, подтвердите их для одного или нескольких виджетов с помощью вызова метода updateAppWidget из AppWidgetManager, передав ему одиночный идентификатор виджета или их массив:

```
appWidgetManager.updateAppWidget(appWidgetIds, views);
```

Стандартный подход к обновлению пользовательского интерфейса виджета заключается в последовательном переборе значений из массива идентификаторов, как показано в листинге 10.6. Это позволяет присваивать разные значения для каждого конкретного виджета, основываясь на настройках или требованиях UI.

Листинг 10.6. Стандартный подход к обновлению виджета

```
final int N = appWidgetIds.length;
// Пройдитесь по виджетам, создавая и применяя
// объект RemoteViews для каждого из них
for (int i = 0; i < N; i++) {
  int appWidgetId = appWidgetIds[i];
  // Создайте новый объект RemoveViews
  RemoteViews views = new RemoteViews(context.getPackageName(),
                                      R.layout.my_widget_layout);

  // TODO Обновлять пользовательский интерфейс виджета с помощью
  // объекта views.

  // Обновите виджет с помощью AppWidgetManager, используя
  // измененный объект RemoveViews.
  appWidgetManager.updateAppWidget(appWidgetId, views);
}
```

Использование RemoteViews внутри обработчика onUpdate из AppWidgetProvider

AppWidgetProvider упрощает взаимодействие с вашим виджетом, передавая объект AppWidgetManager и массив идентификаторов соответствующих виджетов в виде параметров для обработчика onUpdate.

Вы можете использовать подход, описанный выше, без необходимости предварительно получать ссылку на AppWidgetManager или искать значения идентификаторов для нужных виджетов, как показано в листинге 10.7.

Листинг 10.7. Использование RemoteViews внутри обработчика onUpdate из AppWidgetProvider

```
@Override
public void onUpdate(Context context,
                     AppWidgetManager appWidgetManager,
                     int[] appWidgetIds) {
  final int N = appWidgetIds.length;
  for (int i = 0; i < N; i++) {
    int appWidgetId = appWidgetIds[i];
    // Создайте новый объект RemoveViews
    RemoteViews views = new RemoteViews(context.getPackageName(),
                                        R.layout.my_widget_layout);
    // TODO Обновить пользовательский интерфейс.

    // Обновите виджет с помощью AppWidgetManager, используя
    // измененный объект RemoveViews.
    appWidgetManager.updateAppWidget(appWidgetId, views);
  }
}
```

Использование RemoteViews для изменения пользовательского интерфейса

Класс RemoteViews предоставляет различные методы, упрощающие получение доступа к свойствам и методам Представлений, что позволяет изменять их внешний вид.

С помощью наиболее универсальных из них можно вызвать нужный метод из Представления, выполняющегося в другом процессе. Методы из RemoteViews принимают пары параметр — значение. Поддерживаемые сигнатуры включают в себя параметры для каждого простого типа (Boolean, integer, float, а также строки, растровые изображения и параметры для пути URI).

В листинге 10.8 показан пример некоторых поддерживаемых сигнатур.

Листинг 10.8. Использование RemoteViews для изменения пользовательского интерфейса виджета

```
// Установить значение для свойства ImageView.
views.setInt(R.id.widget_image_view, "setImageLevel", 2);
// Показать Курсор в элементе TextView.
views.setBoolean(R.id.widget_text_view, "setCursorVisible", true);
// Передать изображения в элемент ImageButton.
views.setBitmap(R.id.widget_image_button, "setImageBitmap", myBitmap);
```

Класс RemoteViews также содержит набор методов, зависящих от конкретного Представления, чтобы устанавливать значения, применимые к определенному классу, включая TextView, ImageView, ProgressBar и Chronometer.

В листинге 10.9 показан пример использования некоторых этих специфических методов.

Листинг 10.9. Изменение свойств Представления с помощью RemoteViews

```
// Обновить TextView
views.setTextViewText(R.id.widget_text_view, "Updated Text");
views.setTextColor(R.id.widget_text_view, Color.BLUE);
// Обновить ImageView
views.setImageViewBitmap(R.id.widget_image_view, myBitmap);
// Обновить ProgressBar
views.setProgressBar(R.id.widget_progressbar, 100, 50, false);
// Обновить Chronometer
views.setChronometer(R.id.widget_chronometer,
SystemClock.elapsedRealtime(), null, true);
```

Вы также можете показать или скрыть любое Представление, хранящееся в RemoteViews, вызвав метод setViewVisibility, как показано ниже:

```
views.setViewVisibility(R.id.widget_text_view, View.VISIBLE);
```

Как уже было описано в предыдущем разделе, внося изменения в объект RemoteViews, вы должны использовать AppWidgetManager, чтобы применить их к конкретному виджету, как показано в следующем фрагменте:

```
appWidgetManager.updateAppWidget(appWidgetId, views);
```

Добавление интерактивности в виджеты

Вы можете добавить интерактивность в виджеты с помощью RemoteViews, но помните, что типы реакций на пользовательский ввод, которые вам будут доступны, строго ограничены.

Поскольку виджеты работают внутри процесса домашнего экрана, они автоматически наследуют его полномочия. В результате такой политики безопасности интерактивность виджета тщательно контролируется.

Взаимодействия с виджетом, как правило, ограничены двумя возможностями:

- добавление реакции на нажатие для одного или нескольких Представлений внутри разметки;
- изменение пользовательского интерфейса, основанное на переключении текущего выбора.

Примечательно, что нет стандартных способов для ввода текста непосредственно в виджет.

Если вам необходимо получить такую возможность, лучший вариант — добавить реакцию на событие нажатия, при которой отображается Активность, способная принимать пользовательский ввод.

ПРИМЕЧАНИЕ

Один из популярных альтернативных методов — использование элемента ImageView, внешний вид которого «подгоняется» под TextView. Фокус ввода можно получить с помощью механизма изображений, зависящих от выбранного состояния. При нажатии на ImageView запускается частично прозрачная Активность, которая принимает текст, вводимый пользователем.

Использование реакций на нажатие. Наиболее мощный способ, чтобы добавить интерактивность в ваш виджет — это использовать метод setOnClickPendingIntent из объекта RemoteViews.

Это позволит указать Ожидающее намерение, которое сработает в момент, когда пользователь нажмет на определенном Представлении, содержащемся в виджете. Ожидающие намерения (более детально описаны в главе 5) могут использоваться для запуска Активностей и Сервисов или трансляции Намерений.

В листинге 10.10 показано, как назначить Широковещательное намерение для элемента TextView внутри разметки виджета.

Листинг 10.10. Добавление к виджету реакции на нажатие

```
Intent intent = new Intent("com.paad.ACTION_WIDGET_CLICK");
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);
views.setOnClickPendingIntent(R.id.my_text_view, pendingIntent);
```

Используя данную методику, можно добавлять обработчики нажатий для одного или нескольких Представлений, содержащихся в вашем виджете, что может означать поддержку сразу нескольких действий.

Например, стандартный виджет для управления медиапроигрывателем назначает разные Широковещательные намерения для нескольких элементов, обеспечивая управление проигрыванием посредством кнопок play, pause и next.

Внесение изменений в ImageView в зависимости от переключения фокуса. ImageView — один из наиболее гибких типов Представлений, доступных для использования внутри виджетов. Он предоставляет поддержку некоторых базовых видов взаимодействия с пользователем.

Используя ресурс SelectionStateDrawable (описан в главе 3), вы можете создавать ресурсы Drawable. Данный вид ресурсов позволяет отображать разные изображения, основываясь на состоянии Представления, в которое они переданы. С помощью SelectionStateDrawable можно создавать динамический пользовательский интерфейс, который выделяет выбранный пользователем элемент при навигации по виджету.

Отрывок кода в формате XML из листинга 10.11 показывает пример простого ресурса SelectionStateDrawable.

Листинг 10.11. Ресурс SelectionStateDrawable для виджета

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_window_focused="false"
        android:drawable="@drawable/widget_bg_normal"/>
  <item android:state_focused="true"
        android:drawable="@drawable/widget_bg_selected"/>
  <item android:state_pressed="true"
        android:drawable="@drawable/widget_bg_pressed"/>
  <item android:drawable="@drawable/widget_bg_normal"/>
</selector>
```

Ресурсы Drawable, упоминающиеся в данном файле, а также сам файл нужно сохранять в каталоге приложения res/drawable. Далее вы сможете напрямую использовать SelectionStateDrawable в качестве ресурса для ImageView или в виде фона для любого Представления внутри виджета.

Обновление ваших виджетов

Виджеты, как правило, отображаются на домашнем экране, поэтому важно, чтобы они всегда были актуальными. Лучше найти золотую середину между обеспечением этой актуальности и влиянием вашего виджета на системные ресурсы, в частности на время работы батареи.

В следующих разделах описано несколько методик управления интервалами обновления для виджетов.

Использование минимальной частоты обновления

Самый простой, но наиболее ресурсоемкий метод заключается в определении минимальной частоты виджета, которая описывается в файле формата XML, как показано в листинге 10.12 (обновление виджета происходит каждый час):

Листинг 10.12. Установка минимальной частоты обновления для виджета

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:initialLayout="@layout/my_widget_layout"
  android:minWidth="146dp"
  android:minHeight="146dp"
  android:label="My App Widget"
  android:updatePeriodMillis="3600000"
/>
```

Установка этого значения приведет к тому, что устройство будет передавать с указанной частотой объект Intent, вызывающий обновление вашего виджета.

ПРИМЕЧАНИЕ

Чтобы завершить эти обновления, устройство будет выведено из режима ожидания. Это значит, что обновления завершатся даже тогда, когда аппарат бездействует. Потенциально это может привести к перерасходу ценных ресурсов, поэтому важно учитывать последствия, к которым может привести такой подход.

Данная методика должна использоваться для определения абсолютной минимальной частоты, с которой вашему виджету необходимо обновляться, чтобы всегда быть актуальным и полезным. По сути, минимальный ожидаемый интервал между обновлениями должен составлять не меньше одного часа, в идеале обновления должны происходить раз или два в день.

Если ваш виджет нуждается в более частых обновлениях, попробуйте использовать одну из методик, которые описываются в следующих разделах. Чтобы провести обновления динамически, примените модель, основанную на событиях или Намерениях (или более эффективный способ с применением Сигнализации).

Отслеживание Намерений

Поскольку виджеты реализовываются в виде Приемников намерений, вы можете инициировать обновления их состояния и пользовательского интерфейса с помощью регистрации Фильтра намерений для дополнительных действий.

Это динамический подход к обновлению виджетов. Ввместо указания минимальной частоты обновления, что может повлечь излишнюю трату ресурсов батареи, используется более эффективная событийная модель.

В листинге 10.13 приведен пример кода в формате XML, в котором Фильтр намерений добавляется в ту часть манифеста, где описывается виджет, созданный ранее.

Листинг 10.13. Отслеживание передачи Намерений внутри виджета

```
<receiver android:name=".MyAppWidget" android:label="My App Widget">
  <intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>
  <intent-filter>
    <action android:name="com.paad.chapter9.FORCE_WIDGET_UPDATE" />
  </intent-filter>
  <meta-data
    android:name="android.appwidget.provider"
    android:resource="@xml/my_app_widget_info"
  />
</receiver>
```

Отредактировав обработчик onReceive, как показано в листинге 10.14, вы можете отслеживать новое Намерение и использовать его, чтобы обновлять свой виджет.

Листинг 10.14. Обновление виджета, основанное на передаче Намерений

```
public static String FORCE_WIDGET_UPDATE = "com.paad.chapter9.FORCE_
WIDGET_UPDATE";

@Override
public void onReceive(Context context, Intent intent) {
  super.onReceive(context, intent);

  if (FORCE_WIDGET_UPDATE.equals(intent.getAction())) {
    // TODO Обновить пользовательский интерфейс виджета.
  }
}
```

Чтобы инициировать обновление виджета в любой точке вашего приложения, можно передавать Намерение следующим образом:

```
context.sendBroadcast(new Intent(FORCE_WIDGET_UPDATE));
```

Этот способ особенно полезен, когда нужно обеспечить реакцию на системные, пользовательские или программные события, такие как обновление данных, нажатие кнопки внутри виджета, а также изменения в состоянии подключения к сети, уровне заряда батареи или яркости экрана.

Использование механизма Сигнализации

Сигнализацию можно назвать компромиссом между двумя подходами, описанными выше.

Механизм ее, подробно рассмотренный в главе 9, предоставляет гибкий способ планирования регулярных событий внутри вашего приложения. С его помощью вы можете осуществлять регулярные действия с заданным интервалом, используя объекты Intent для запуска ваших обновлений.

Использование Сигнализации для обновления виджетов похоже на использование ранее описанной модели, в основе которой лежат Намерения. Добавьте новый Фильтр намерений в секцию вашего виджета в манифесте и переопределите обработчик onReceive, чтобы определять Намерение, которое его вызвало. Используйте AlarmManager внутри вашего приложения, чтобы создать Сигнализацию, которая активизирует Намерения при осуществлении определенного действия.

Благодаря гибкости механизм Сигнализации имеет преимущество перед простой установкой минимальной частоты обновления.

Сигнализация при необходимости может выводить устройство из режима ожидания, что требует от разработчика такого же пристального внимания к расходу ресурсов батареи.

Отличие в том, что, устанавливая режимы RTC или ELAPSED_REALTIME при создании Сигнализации, вы можете настроить ее таким образом, чтобы

она срабатывала только в том случае, если устройство не нуждается в «пробуждении»:

```
larmManager.setRepeating(AlarmManager.ELAPSED_REALTIME,
                         AlarmManager.INTERVAL_HOUR,
                         AlarmManager.INTERVAL_HOUR,
                         pi);
```

Используя данный подход, вы гарантируете, что ваш виджет остается актуальным в тот момент, когда он видим на экране, при этом не расходуете ценные ресурсы тогда, когда экран выключен.

Если ваш виджет должен обновляться даже тогда, когда устройство находится в режиме ожидания, можете оптимизировать этот процесс с помощью параметра неточного повторения, как показано ниже:

```
String alarmService = Context.ALARM_SERVICE;
AlarmManager alarmManager = (AlarmManager)getSystemService(alarmService);

Intent intent = new Intent(MyAppWidget.FORCE_WIDGET_UPDATE);
PendingIntent pi = PendingIntent.getBroadcast(this,
                                              0,
                                              intent,
                                              0);

alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                                 AlarmManager.INTERVAL_HALF_DAY,
                                 AlarmManager.INTERVAL_HALF_DAY,
                                 pi);
```

Как уже описывалось в главе 9, при неточном повторении Сигнализация оптимизирует свои срабатывания, сдвигая все запланированные события на один и тот же момент времени. Благодаря этому устройство «проснется» только один раз, вместо того чтобы неоднократно выходить из режима ожидания в течение нескольких минут.

Создание и использование Активности для настройки виджета

В некоторых случаях виджет может быть намного полезней, если пользователь получит возможность настраивать его содержимое и внешний вид. Это важно, особенно учитывая тот факт, что на домашний экран можно добавить сразу несколько экземпляров одного и того же виджета.

Активность, необходимая для настройки виджета, должна немедленно загружаться при добавлении этого виджета на домашний экран. Это может быть любая Активность в рамках вашего приложения при условии, что она имеет Фильтр намерений для действия APPWIDGET_CONFIGURE, как показано ниже:

```
<activity android:name=". MyWidgetConfigurationActivity">
  <intent-filter>
```

```
    <action android:name="android.apwidget.action.APPWIDGET_CONFIGURE"/>
  </intent-filter>
</activity>
```

Она также должна возвращать результирующее Намерение, которое включает параметр extra, содержащий значение идентификатора виджета для константы EXTRA_APPWIDGET_ID. Этот параметр должен быть добавлен в Намерение, которое запускает Активность.

```
Intent result = new Intent();
result.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
setResult(RESULT_OK, result);
finish();
```

Чтобы назначить окончательный вариант Активности для виджета, нужно добавить ее в файл конфигурации виджета с помощью атрибута configure. Активность должна быть представлена в виде полного имени пакета, как показано ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:initialLayout="@layout/my_widget_layout"
  android:minWidth="146dp"
  android:minHeight="146dp"
  android:label="My App Widget"
  android:updatePeriodMillis="3600000"
  android:configure="com.paad.chapter9.MyWidgetConfigurationActivity"
/>
```

Создание виджета для приложения Earthquake

Выполнив следующие инструкции, вы создадите новый виджет для домашнего экрана, который будет отображать информацию о последних зафиксированных землетрясениях. Пользовательский интерфейс этого виджета максимально прост (это «побочный эффект» стремления сделать примеры как можно более лаконичными). Обратите внимание, что данный виджет создается без учета рекомендаций о визуальном стиле, речь о которых шла ранее в этой главе.

После завершения работ над виджетом и добавления его на домашний экран выглядеть он будет так, как показано на рис. 10.2.

Используя комбинацию методик обновления, описанных выше, этот виджет отслеживает Широковещательные намерения, которые должны оповещать о том, что обновление было завершено, также для виджета установлена гарантированная минимальная частота обновления (один раз в день).

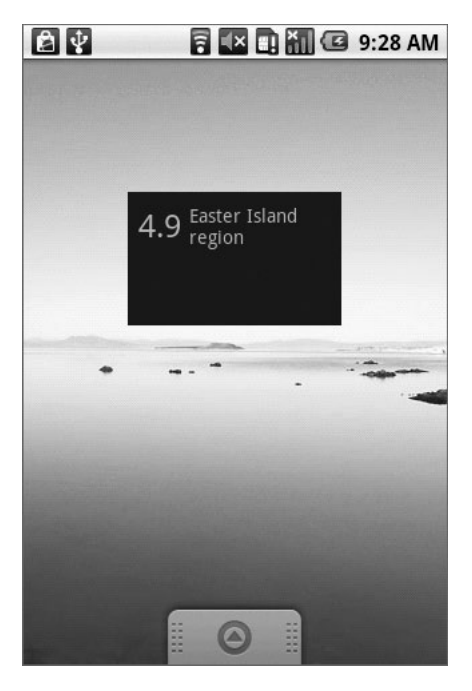


Рис. 10.2.

Следующий код дополняет приложение Earthquake, с которым вы последний раз имели дело в главе 8.

1. Начните с создания разметки для пользовательского интерфейса виджета в виде ресурса XML. Используйте LinearLayout для настройки отображения элемента TextView, который выводит магнитуду и местоположение землетрясения:

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#F111"
  android:padding="5sp">
  <TextView
    android:id="@+id/widget_magnitude"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:textSize="24sp"
    android:padding="3dp"
  />
  <TextView
    android:id="@+id/widget_details"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:textSize="14sp"
    android:padding="3dp"
  />
</LinearLayout>
```

2. Создайте заглушку для нового класса EarthquakeWidget, потомка AppWidgetProvider. Вы вернетесь к нему позже, чтобы реализовать механизм обновления вашего виджета, обеспечивая его самыми свежими данными о землетрясениях.

```
package com.paad.earthquake;

import android.widget.RemoteViews;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;

public class EarthquakeWidget extends AppWidgetProvider {
}
```

3. Создайте новый файл для описания виджета `quake_widget_info.xml`, и поместите его в каталог res/xml. Установите минимальную частоту обновления до 24 часов, укажите, что виджет будет занимать две ячейки в ширину и одну в высоту 146 × 74 dp. Используйте разметку, которую вы создали на шаге 1 как исходную.

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:initialLayout="@layout/quake_widget"
  android:minWidth="146dp" android:minHeight="74dp"
  android:label="Last Earthquake"
  android:updatePeriodMillis="86400000"
/>
```

4. Добавьте виджет в манифест приложения, включив в него ссылку на ресурс с описанием, который вы создали на предыдущем шаге. Зарегистрируйте Фильтр намерений для обновления виджета.

```
<receiver android:name="EarthquakeWidget" android:label="Last
Earthquake">
  <intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>
  <meta-data
    android:name="android.appwidget.provider"
    android:resource="@xml/earthquake_widget_info"
  />
</receiver>
```

Теперь ваш виджет настроен и готов для размещения на домашнем экране. Сейчас вам необходимо отредактировать класс EarthquakeWidget, созданный на шаге 2, чтобы реализовать механизм обновления, обеспечивая виджет самыми свежими данными о землетрясениях.

5. Создайте два новых перегруженных метода updateQuake внутри класса виджета.

   5.1. Первый перегруженный метод должен принимать в качестве параметров объекты AppWidgetManager и Context, а также массив, в котором будут храниться идентификаторы виджета. Позже вы расширите эту заглушку, чтобы обновлять внешний вид виджета с помощью объекта RemoteViews.

```
public void updateQuake(Context context,
                        AppWidgetManager appWidgetManager,
                        int[] appWidgetIds) {
}
```

   5.2. Второй метод-заглушка должен принимать только объект Context, используя его для получения экземпляра AppWidgetManager. Далее применяйте AppWidgetManager, чтобы найти идентификаторы активных виджетов приложения Earthquake, передавая их в качестве параметров для метода, который вы создали на шаге 5.1.

```
public void updateQuake(Context context) {
  ComponentName thisWidget = new ComponentName(context,
                                        EarthquakeWidget.class);
  AppWidgetManager appWidgetManager =
    AppWidgetManager.getInstance(context);
  int[] appWidgetIds = appWidgetManager.getAppWidgetIds(thisWidget);
  updateQuake(context, appWidgetManager, appWidgetIds);
}
```

   5.3. Внутри заглушки updateQuake из пункта 5.1 используйте Источник данных EarthquakeProvider, созданный в главе 6, чтобы получать последние землетрясения и извлекать их магнитуду и местоположение:

```
public void updateQuake(Context context,
                        AppWidgetManager appWidgetManager,
                        int[] appWidgetIds) {

  Cursor lastEarthquake;
  ContentResolver cr = context.getContentResolver();
  lastEarthquake = cr.query(EarthquakeProvider.CONTENT_URI,
                            null, null, null, null);

  String magnitude = "--";
  String details = "-- None --";

  if (lastEarthquake != null) {
    try {
```

```
      if (lastEarthquake.moveToFirst()) {
        magnitude = lastEarthquake.getString(EarthquakeProvider.
MAGNITUDE_COLUMN);
        details =
lastEarthquake.getString(EarthquakeProvider.DETAILS_COLUMN);
      }
    }
    finally {
      lastEarthquake.close();
    }
  }
}
```

5.4. Создайте новый объект RemoteViews, чтобы задать текст, отображаемый элементом TextView внутри виджета. Это нужно для того, чтобы показывать магнитуду и местоположение последнего землетрясения:

```
public void updateQuake(Context context,
                        AppWidgetManager appWidgetManager,
                        int[] appWidgetIds) {

  Cursor lastEarthquake;
  ContentResolver cr = context.getContentResolver();
  lastEarthquake = cr.query(EarthquakeProvider.CONTENT_URI,
                            null, null, null, null);

  String magnitude = "--";
  String details = "-- None --";

  if (lastEarthquake != null) {
    try {
      if (lastEarthquake.moveToFirst()) {
        magnitude =
lastEarthquake.getString(EarthquakeProvider.MAGNITUDE_COLUMN);
        details =
lastEarthquake.getString(EarthquakeProvider.DETAILS_COLUMN);
      }
    }
    finally {
      lastEarthquake.close();
    }
  }

  final int N = appWidgetIds.length;
  for (int i = 0; i < N; i++) {
    int appWidgetId = appWidgetIds[i];
    RemoteViews views = new RemoteViews(context.getPackageName(),
                                        R.layout.quake_widget);
    views.setTextViewText(R.id.widget_magnitude, magnitude);
    views.setTextViewText(R.id.widget_details, details);
    appWidgetManager.updateAppWidget(appWidgetId, views);
  }
}
```

6. Переопределите обработчик onUpdate, чтобы вызвать метод update-Quake:

```
@Override
public void onUpdate(Context context,
                     AppWidgetManager appWidgetManager,
                     int[] appWidgetIds) {
  updateQuake(context, appWidgetManager, appWidgetIds);
}
```

   Теперь ваш виджет готов к использованию и начнет обновлять информацию о землетрясениях каждые 24 часа, как только попадет на домашний экран.

7. Доработайте виджет, чтобы он обновлялся всякий раз, когда Сервис Earthquake, созданный вами в главе 8, вносит изменения в базу данных с землетрясениями.

   7.1. Начните с редактирования метода doRefreshEarthquakes из класса EarthquakeService, чтобы передавать Намерение после завершения обновлений.

```
public static String QUAKES_REFRESHED =
  "com.paad.earthquake.QUAKES_REFRESHED";
public void doRefreshEarthquakes() {
  [ ... Ранее написанный код ... ]
  sendBroadcast(new Intent(QUAKES_REFRESHED));
}
```

   7.2. Переопределите метод onReceive в классе EarthquakeWidget, но не забудьте сперва вызвать одноименный метод родительского класса, чтобы по-прежнему срабатывали стандартные обработчики событий для виджета:

```
@Override
public void onReceive(Context context, Intent intent){
  super.onReceive(context, intent);
}
```

   7.3. Проверьте, имеет ли переданное действие из описанного выше пункта 7.1 тип QUAKES_REFRESHED, и вызовите при его получении метод updateQuakes:

```
@Override
public void onReceive(Context context, Intent intent){
  super.onReceive(context, intent);


  if (intent.getAction().equals(EarthquakeService.QUAKES_REFRESHED))
    updateQuake(context);
}
```

   7.4. В завершение добавьте Фильтр намерений для этого действия в раздел манифеста, описывающего виджет:

```
<receiver android:name="EarthquakeWidget" android:label="LastEarthquake">
  <intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>
  <intent-filter>
    <action android:name="com.paad.earthquake.QUAKES_REFRESHED" />
  </intent-filter>
  <meta-data
    android:name="android.appwidget.provider"
    android:resource="@xml/earthquake_widget_info"
  />
</receiver>
```

Теперь ваш виджет будет обновляться один раз в день, а также каждый раз, когда Сервис Earthquake ведет поиск нового землетрясения.

Чтобы улучшить виджет, подумайте, каким образом можно использовать объект Drawable со слоями внутри ImageView для наглядного представления значения магнитуды отображаемого землетрясения. На рис. 10.3 показан один из вариантов.



Рис. 10.3.

Знакомство с Живыми каталогами

Живые каталоги — это уникальный и мощный механизм, с помощью которого ваши приложения могут выводить информацию из своих Источников

данных прямо на домашнем экране. Они предоставляют динамические ссылки на данные, которые хранит ваше приложение.

После добавления на домашний экран Живой каталог выглядит, как обычный значок для запуска приложений. При нажатии на нем открывается меню Живого каталога, как показано на рис. 10.4. В данном случае изображен Живой каталог со списком избранных контактов, открытый на домашнем экране Android.

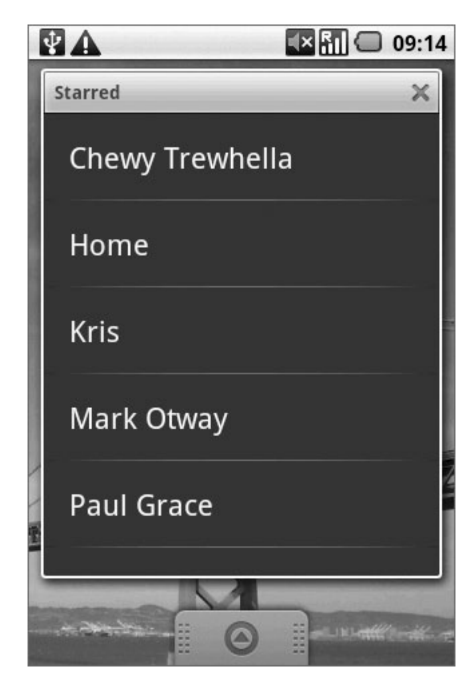


Рис. 10.4.

ПРИМЕЧАНИЕ

Чтобы добавить Живой каталог на домашний экран, сделайте продолжительное нажатие на свободном участке и выберите раздел Folders. Откроется список доступных Живых каталогов. Нажмите на один из них, чтобы выбрать. Чтобы открыть Живой каталог, нужно просто на него нажать; передвигать его можно с помощью продолжительного нажатия.

Создание Живых каталогов

Живые каталоги — комбинация двух элементов: Источника данных, который возвращает информацию в стандартном формате, необходимую для заполнения Живого каталога, и Активности, которая возвращает Намерение, используемое для генерации Живого каталога.

Чтобы создать новый Живой каталог, нужно определить:

- Активность, ответственную за создание и настройку Живого каталога, она будет создавать и возвращать специального вида Намерение;
- Источник данных, предоставляющий элементы, которые будут отображаться с помощью корректных имен столбцов.

Каждый элемент в Живом каталоге может быть представлен в виде трех частей: значка, названия и описания.

Источники данных для Живых каталогов

Любой Источник данных может предоставлять информацию, отображающуюся в Живом каталоге. В них используется стандартный набор имен столбцов.

- LiveFolders._ID. Уникальный идентификатор, который указывает на то, какой именно элемент был выбран, если пользователь сделал нажатие в меню Живого каталога.
- LiveFolders.NAME. Заголовочный текст, отображаемый с помощью большого шрифта. Этот столбец — единственный обязательный элемент.
- LiveFolders.DESCRIPTION. Более длинное поле для описания, отображается с помощью меньшего шрифта под заголовком.
- LiveFolders.IMAGE. Изображение, размещающееся в левой части элемента.

При отображении Живой каталог использует имена этих столбцов для извлечения информации из Источника данных.

Вместо внесения изменений в Источник данных, чтобы он соответствовал требованиям Живых каталогов, лучше применить проекцию, которая свяжет необходимые имена столбцов с уже существующими столбцами Источника данных, как показано в листинге 10.15.

Листинг 10.15. Создание проекции для поддержки Живого каталога

```
final HashMap<String, String> liveFolderProjection =
  new HashMap<String, String>();
liveFolderProjection.put(LiveFolders._ID,
                         KEY_ID + " AS " +
                         LiveFolders._ID);
liveFolderProjection.put(LiveFolders.NAME,
                         KEY_NAME_COLUMN + " AS " +
                         LiveFolders.NAME);
liveFolderProjection.put(LiveFolders.DESCRIPTION,
                         KEY_DESCRIPTION_COLUMN + " AS " +
                         LiveFolders.DESCRIPTION);
```

Продолжение ⇨

Листинг 10.15 (продолжение)

```
liveFolderProjection.put(LiveFolders.IMAGE,
                         KEY_IMAGE_COLUMN + " AS " +
                         LiveFolders.IMAGE);

SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
qb.setTables(MY_TABLES);
qb.setProjectionMap(LIVE_FOLDER_PROJECTION);
```

Обязательными столбцами сичтаются только _ID и NAME, при необходимости изображение и описание можно не использовать.

Активность для создания Живого каталога

Живой каталог создается с помощью Намерения, возвращаемого в виде результата работы Активности. Свойство Намерение data содержит путь URI к информации, предоставляемой Источником данных (с применением соответствующей проекции). Свойства extra отвечают за хранение настроек, таких как режим отображения, значок, имя каталога.

В листинге 10.16 показан переопределенный метод onCreate из Активности, которая будет использована для создания Живого каталога.

Намерение, которое определяет Живой каталог, создается и возвращается в виде результата выполнения Активности, прежде чем она закроется с помощью метода finish.

Листинг 10.16. Активность для создания Живого каталога

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  String action = getIntent().getAction();
  if (LiveFolders.ACTION_CREATE_LIVE_FOLDER.equals(action)) {
    Intent intent = new Intent();
    intent.setData(EarthquakeProvider.LIVE_FOLDER_URI);
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_BASE_INTENT,
                    new Intent(Intent.ACTION_VIEW,
                               EarthquakeProvider.CONTENT_URI));
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_DISPLAY_MODE,
                    LiveFolders.DISPLAY_MODE_LIST);
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_ICON,
    Intent.ShortcutIconResource.fromContext(context,
                                             R.drawable.icon));
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_NAME, "Earthquakes");
    setResult(RESULT_OK, createLiveFolderIntent(this));
  }
  else
    setResult(RESULT_CANCELED);
  finish();
}
```

Наряду со стандартными значениями настроек вы можете добавить параметр LiveFolders.EXTRA_LIVE_FOLDER_BASE_INTENT, чтобы указать базовое Намерение, которое сработает, если Живой каталог выбран из меню.

При выборе элемента Живой каталог вызывает метод showActivity, передавая ему Намерение с параметром data, который содержит базовый путь URI к Живому каталогу и добавленный к нему идентификатор выбранного элемента.

При добавлении Активности вашего Живого каталога в манифест приложения необходимо также включить в него Фильтр намерений для действия CREATE_LIVE_FOLDER, как показано в листинге 10.17.

Листинг 10.17. Добавление Фильтра намерений для Живого каталога

```
<activity android:name=".MyLiveFolder "
          android:label="My Live Folder">
  <intent-filter>
    <action android:name="android.intent.action.CREATE_LIVE_FOLDER"/>
  </intent-filter>
</activity>
```

Создание Живого каталога для приложения Earthquake

В следующем примере вы опять расширите возможности приложения Earthquake, но на этот раз с помощью Живого каталога, который отображает магнитуду и местоположение каждого землетрясения.

1. Начните с редактирования класса EarthquakeProvider. Создайте новое статическое поле с путем URI, которое будет использовано для возвращения элементов Живого каталога.

```
public static final Uri LIVE_FOLDER_URI =
  Uri.parse("content://com.paad.provider.earthquake/live_folder");
```

2. Измените объект uriMatcher и метод getType, чтобы иметь возможность обрабатывать запросы к новому URI.

```
private static final int LIVE_FOLDER = 3;

static {
  uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
  uriMatcher.addURI("com.paad.provider.Earthquake", "earthquakes",
QUAKES);
  uriMatcher.addURI("com.paad.provider.Earthquake", "earthquakes/#",
QUAKE_ID);
  uriMatcher.addURI("com.paad.provider.Earthquake", "live_folder", LIVE_
FOLDER);
}

@Override
public String getType(Uri uri) {
  switch (uriMatcher.match(uri)) {
    case QUAKES|LIVE_FOLDER :
      return "vnd.android.cursor.dir/vnd.paad.earthquake";
    case QUAKE_ID: return "vnd.android.cursor.item/vnd.paad.earthquake";
    default: throw new IllegalArgumentException("Unsupported URI: " + uri);
  }
}
```

3. Создайте новый ассоциативный массив HashMap, который будет хранить определение проекции для Живого каталога. Он должен возвращать информацию о магнитуде и местоположении, а также столбцы с описанием и именем.

```
static final HashMap<String, String> LIVE_FOLDER_PROJECTION;
static {
  LIVE_FOLDER_PROJECTION = new HashMap<String, String>();
  LIVE_FOLDER_PROJECTION.put(LiveFolders._ID,
                             KEY_ID + " AS " + LiveFolders._ID);
  LIVE_FOLDER_PROJECTION.put(LiveFolders.NAME,
                             KEY_DETAILS + " AS " + LiveFolders.NAME);
  LIVE_FOLDER_PROJECTION.put(LiveFolders.DESCRIPTION,
                             KEY_DATE + " AS " + LiveFolders.DESCRIPTION);
}
```

4. Отредактируйте метод query, чтобы применить ассоциативный массив из предыдущего пункта при запросе, который направлен к Живому каталогу.

```
@Override
public Cursor query(Uri uri,
                    String[] projection,
                    String selection,
                    String[] selectionArgs,
                    String sort) {

  SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
  qb.setTables(EARTHQUAKE_TABLE);

  switch (uriMatcher.match(uri)) {
    case QUAKE_ID :
      qb.appendWhere(KEY_ID + "=" + uri.getPathSegments().get(1));
      break;
    case LIVE_FOLDER : qb.setProjectionMap(LIVE_FOLDER_PROJECTION);
                       break;
    default : break;
  }
  [ ... ранее написанный код ... ]
}
```

5. Создайте новый класс EarthquakeLiveFolders, который будет содержать статическую Активность EarthquakeLiveFolder.

```
package com.paad.earthquake;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.provider.LiveFolders;

public class EarthquakeLiveFolders extends Activity {
```

```
  public static class EarthquakeLiveFolder extends Activity {
  }
}
```

6. Добавьте новый метод, который определяет Намерение, нужное для создания Живого каталога. Оно должно использовать путь URI, созданный вами на шаге 1, устанавливать режим отображения для списка, определять значок и заголовочный текст. Установите также базовое Намерение для запроса к единичной записи в Источнике данных EarthquakeProvider.

```
private static Intent createLiveFolderIntent(Context context) {
  Intent intent = new Intent();
  intent.setData(EarthquakeProvider.LIVE_FOLDER_URI);
  intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_BASE_INTENT,
                  new Intent(Intent.ACTION_VIEW,
                             EarthquakeProvider.CONTENT_URI));
  intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_DISPLAY_MODE,
                  LiveFolders.DISPLAY_MODE_LIST);
  intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_ICON,
                  Intent.ShortcutIconResource.fromContext(context,
                                                           R.drawable.
icon));
  intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_NAME, "Earthquakes");
  return intent;
}
```

7. Переопределите метод onCreate для класса EarthquakeLiveFolder, чтобы возвращать Намерение, созданное на шаге 6.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  String action = getIntent().getAction();
  if (LiveFolders.ACTION_CREATE_LIVE_FOLDER.equals(action))
    setResult(RESULT_OK, createLiveFolderIntent(this));
  else
    setResult(RESULT_CANCELED);
  finish();
}
```

8. Добавьте Активность EarthquakeLiveFolder в манифест приложения, включив Фильтр намерений для действия android.intent.action.CREATE_LIVE_FOLDER.

```
<activity android:name=".EarthquakeLiveFolders$EarthquakeLiveFolder"
          android:label="All Earthquakes">
  <intent-filter>
    <action android:name="android.intent.action.CREATE_LIVE_FOLDER"/>
  </intent-filter>
</activity>
```

На рис. 10.5 показан Живой каталог с землетрясениями, открытый на домашнем экране.

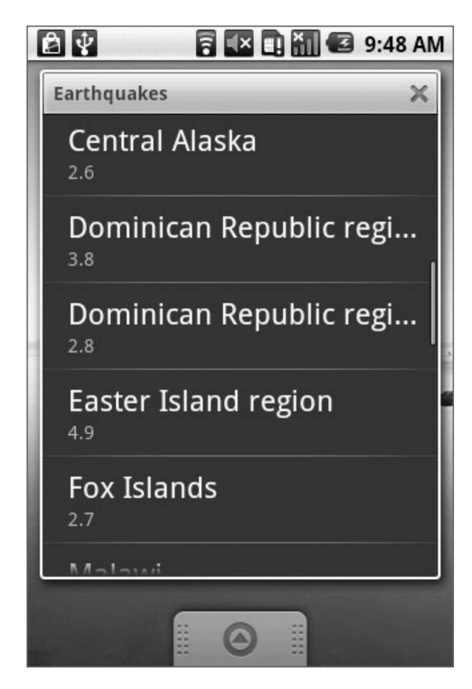


Рис. 10.5.

Вы можете дополнить этот пример, выводя на карте конкретное землетрясение, выбранное из списка.

Начните с добавления Фильтра намерений к Активности EarthquakeMap. Он будет отслеживать действия с Источником данных EarthquakeProvider, описанным с помощью объекта Intent, который был создан на шаге 6. Потом сделайте так, чтобы Активность могла получать местоположение выбранного землетрясения и прокручивать карту к этой точке.

Виджет быстрого поиска и добавление поисковых возможностей в свое приложение

Для приложений, которые предоставляют большие базы данных и хранят существенные объемы информации, поисковые возможности становятся все более значимыми.

Android включает фреймворк для упрощения процесса поиска внутри Источников данных и для вывода полученных результатов. В этом разделе вы узнаете, как в собственное приложение можно добавить поисковые возможности.

Добавление в приложение поисковых возможностей

Большинство устройств под управлением Android оснащены аппаратной кнопкой поиска. Используя данный фреймворк, можно задействовать специфические для вашего приложения поисковые возможности, если пользователь нажал эту кнопку. Поисковая строка будет динамически отображать результаты по мере ввода поискового запроса.

Создание Активности для поиска

Чтобы в вашем приложении можно было вести поиск, необходимо создать Активность, которая станет инициировать и отображать процесс поиска.

Сперва нужно создать новый поисковый ресурс с метаданными в формате XML и разместить его в каталоге res/xml. Этот файл, показанный в листинге 10.18, описывает полномочия Источника данных, в котором вы планируете вести поиск, а также действие, которое будет вызвано, если пользователь нажмет на поисковый результат.

Листинг 10.18. Описание поисковых метаданных для приложения

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
  android:label="@string/app_name"
  android:searchSuggestAuthority="myauthority"
  android:searchSuggestIntentAction="android.intent.action.VIEW">
</searchable>
```

Далее нужно создать Активность, которая задействуется для вывода результатов поиска. Как правило, в этой роли выступает простая Активность, основанная на элементе ListView, но вы при необходимости можете разработать любой пользовательский интерфейс. На примере листинга 10.19 добавьте внутрь тег `<meta-data>`, который имеет атрибут `name` со значением android.app.searchable и атрибут `resource`, ссылающийся на ресурс, созданный в листинге 10.18.

Вы также должны добавить Фильтр намерений, зарегистрированный для действия android.intent.action.SEARCH, и категорию DEFAULT.

Листинг 10.19. Регистрация Активности для отображения результатов поиска

```
<activity android:name=".EarthquakeSearch" android:label="Earthquake
Search">
  <intent-filter>
    <action android:name="android.intent.action.SEARCH" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
  <meta-data
    android:name="android.app.searchable"
    android:resource="@xml/searchable"
  />
</activity>
```

Поисковый запрос, который приведет к отображению данной Активности, будет возвращен внутрь Намерения с помощью параметра extra под названием SearchMananger.USER_QUERY, как показано ниже:

```
String searchTerm = getIntent().getStringExtra(SearchManager.USER_QUERY);
```

На деле лучше придерживаться единого формата вывода поисковых результатов в приложении. Чтобы установить Активность в качестве источника поисковых результатов по умолчанию, необходимо добавить новый тег `<meta-data>` в манифест приложения, как показано в листинге 10.20.

Присвойте атрибуту `name` значение android.app.default_searchable и укажите вашу Активность для вывода поисковых результатов, используя атрибут `value`.

Листинг 10.20. Установка Активности, которая по умолчанию выводит результаты поиска для приложения

```
<meta-data
  android:name="android.app.default_searchable"
  android:value=".EarthquakeSearch"
/>
```

Реакция на поисковые запросы из Источника данных

Активность, описанная в предыдущем разделе, может быть использована как для выполнения поиска, так и для вывода поисковых результатов в приложении. Чтобы получить информацию для вывода ее на экран, нужно создать (или изменить) Источник данных, который будет обрабатывать поисковые запросы и возвращать результаты.

Чтобы воспользоваться поисковым фреймворком Android, вашему Источнику данных необходимо поддерживать специфические пути URI. В листинге 10.21 показан объект UriMatcher, который сравнивает запрашиваемый путь с известными поисковыми запросами, представленными в виде значений URI.

Листинг 10.21. Распознавание поисковых запросов в Источнике данных

```
private static int SEARCH = 1;

static {
  uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
  uriMatcher.addURI("com.paad.provider.earthquake", "earthquakes",
QUAKES);
  uriMatcher.addURI("com.paad.provider.earthquake", "earthquakes/#",
QUAKE_ID);
  uriMatcher.addURI("com.paad.provider.earthquake",
    SearchManager.SUGGEST_URI_PATH_QUERY, SEARCH);
  uriMatcher.addURI("com.paad.provider.earthquake",
    SearchManager.SUGGEST_URI_PATH_QUERY + "/*", SEARCH);
  uriMatcher.addURI("com.paad.provider.earthquake",
```

```
      SearchManager.SUGGEST_URI_PATH_SHORTCUT, SEARCH);
    uriMatcher.addURI("com.paad.provider.earthquake",
      SearchManager.SUGGEST_URI_PATH_SHORTCUT + "/*", SEARCH);
}
```

Используйте этот же шаблон URI внутри Источника данных, чтобы возвращать соответствующие типы MIME для поисковых запросов, как показано в листинге 10.22. Результаты поиска должны возвращаться в виде SearchManager.SUGGEST_MIME_TYPE, чтобы поддерживать подсказки во время ввода.

Листинг 10.22. Возвращение корректного типа MIME для результатов поиска

```
@Override
public String getType(Uri uri) {
  switch (uriMatcher.match(uri)) {
    case QUAKES : return "vnd.android.cursor.dir/vnd.paad.earthquake";
    case QUAKE_ID: return "vnd.android.cursor.item/vnd.paad.earthquake";
    case SEARCH : return SearchManager.SUGGEST_MIME_TYPE;
    default: throw new IllegalArgumentException("Unsupported URI: " + uri);
  }
}
```

Объект UriMatcher также может быть применен внутри метода query. Если фиксируется входящий поисковый запрос, вы можете получить его условия, изучив последний сегмент запрашиваемого пути URI.

```
uri.getPathSegments().get(1);
```

Чтобы вернуть результаты поиска, которые могут быть выведены с помощью поискового фреймворка Android, нужно создать и применить проекцию, приводящую имена столбцов в формат, поддерживаемый объектом SearchManager. Этот объект включает статические константы вида SUGGEST_COLUMN_*, которые могут использоваться в проекции.

Два столбца обязательны: SUGGEST_COLUMN_TEXT_1 — показывает текст результата поиска, и id_ — хранит уникальный идентификатор строки.

В листинге 10.23 показан каркас для создания и применения проекции внутри запроса, он возвращает объект Cursor, соответствующий результатам поиска.

Листинг 10.23. Возвращение результатов поиска из метода query

```
private static final HashMap<String, String> SEARCH_PROJECTION_MAP;
static {
  SEARCH_PROJECTION_MAP = new HashMap<String, String>();
  SEARCH_PROJECTION_MAP.put(SearchManager.SUGGEST_COLUMN_TEXT_1,
                            KEY_SEARCH_COLUMN + " AS " +
                            SearchManager.SUGGEST_COLUMN_TEXT_1);
```

Продолжение ⇗

Листинг 10.23 (продолжение)

```
    SEARCH_PROJECTION_MAP.put("_id", KEY_ID + " AS " + "_id");
  }

  @Override
  public Cursor query(Uri uri, String[] projection, String selection,
  String[]
                      selectionArgs, String sort) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(MY_TABLE);

    switch (uriMatcher.match(uri)) {
      case SINGLE_ID:
        qb.appendWhere(KEY_ID + "=" + uri.getPathSegments().get(1));
        break;
      case SEARCH   :  qb.appendWhere(KEY_SEARCH_COLUMN + " LIKE \"%" +
                                  uri.getPathSegments().get(1) + "%\"");
                       qb.setProjectionMap(SEARCH_PROJECTION_MAP);
                       break;
      default       : break;
    }

    Cursor c = qb.query(MyDB,
                        projection,
                        selection, selectionArgs,
                        null, null, orderBy);

    return c;
  }
```

Доставка поисковых результатов в Виджет быстрого поиска

В Android 1.6 (API Level 4) появилась возможность предоставлять поисковые результаты из вашего приложения с помощью универсального Виджета быстрого поиска.

Этот виджет расположен на видном месте домашнего экрана, и пользователь может загрузить его в любой момент, нажав аппаратную кнопку поиска. При передаче поисковых результатов из вашего приложения с помощью данного механизма вы даете пользователю еще одну дополнительную точку входа в свое приложение.

Чтобы получить возможность доставлять свои поисковые результаты в Виджет быстрого поиска, необходимо сперва реализовать поисковые возможности внутри своего приложения, как было описано в предыдущем разделе.

Для того чтобы сделать ваши результаты доступными глобально, отредактируйте файл `searchable.xml`, который описывает поисковые ме-

таданные приложения, и добавьте два новых атрибута, как показано в листинге 10.24:

- `searchSettingsDescription` — используется для описания ваших поисковых результатов в меню настроек;
- `includeInGlobalSearch` — установите этому атрибуту значение true, чтобы передавать поисковые результаты в Виджет быстрого поиска.

Листинг 10.24. Добавление поисковых результатов в Виджет быстрого поиска

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
  android:label="@string/app_name"
  android:searchSettingsDescription="@string/app_name"
  android:includeInGlobalSearch="true"
  android:searchSuggestAuthority="com.paad.provider.earthquake"
  android:searchSuggestIntentAction="android.intent.action.VIEW">
</searchable>
```

Обратите внимание, что ваши поисковые результаты не станут автоматически и напрямую доставляться в Виджет быстрого поиска. Во избежание возможных злоупотреблений добавление новых источников поиска требует явных действий от самого пользователя.

Чтобы добавить новый поисковый источник для Виджета быстрого поиска, используйте системные настройки. Перейдите в Settings ▸ Search ▸ Searchable Items и отметьте те источники, которые хотите включить.

ПРИМЕЧАНИЕ

Поскольку добавление результатов в Виджет быстрого поиска зависит от пользовательских настроек, постарайтесь предусмотреть уведомления о новой доступной функциональности.

Добавление поисковых возможностей в приложение Earthquake

В следующем примере[1] вы добавите возможность поиска в проект Earthquake, а также сделаете поисковые результаты доступными через Виджет быстрого поиска на домашнем экране.

1. Начните с добавления двух новых строковых ресурсов в файл `strings.xml`, который находится в каталоге res/values. Один будет содержит имя для определения результатов поиска по землетрясениям, другой станет выступать в качестве описания.

```
<string name="search_label">Earthquakes</string>
<string name="search_description">Earthquake locations</string>
```

[1] Все фрагменты кода в этом примере — часть проекта Earthquake из главы 10, их можно загрузить с сайта Wrox.com.

2. Создайте новый каталог для ресурсов в формате XML — res/xml. Добавьте туда новый файл `searchable.xml`, в котором будут описываться метаданные для источника поиска в приложении Earthquake. Укажите строки из предыдущего пункта в качестве атрибутов `label` и `searchSettingsDescription`. Определите полномочия для Источника данных Earthquake и присвойте атрибуту `includeInGlobalSearch` значение true.

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
  android:label="@string/app_name"
  android:searchSettingsDescription="@string/app_name"
  android:includeInGlobalSearch="true"
  android:searchSuggestAuthority="com.paad.provider.earthquake"
  android:searchSuggestIntentAction="android.intent.action.VIEW">
</searchable>
```

3. Перейдите к классу EarthquakeProvider. Начните с добавления новой статической константы SEARCH_URI, которую вы сможете использовать для поиска внутри приложения.

```
public static final Uri SEARCH_URI =
  Uri.parse("content://com.paad.provider.earthquake/" +
            SearchManager.SUGGEST_URI_PATH_QUERY);
```

4. Создайте новую проекцию, которая потребуется для доставки поисковых результатов.

```
private static final HashMap<String, String> SEARCH_PROJECTION_MAP;
static {
  SEARCH_PROJECTION_MAP = new HashMap<String, String>();
  SEARCH_PROJECTION_MAP.put(SearchManager.SUGGEST_COLUMN_TEXT_1,
KEY_DETAILS +
    " AS " + SearchManager.SUGGEST_COLUMN_TEXT_1);
  SEARCH_PROJECTION_MAP.put("_id", KEY_ID +
    " AS " + "_id");
}
```

5. Отредактируйте объект UriMatcher, чтобы добавить поддержку поисковых запросов.

```
private static int SEARCH = 3;

static {
  uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
  uriMatcher.addURI("com.paad.provider.earthquake", "earthquakes",
QUAKES);
  uriMatcher.addURI("com.paad.provider.earthquake", "earthquakes/#",
QUAKE_ID);
  uriMatcher.addURI("com.paad.provider.earthquake",
    SearchManager.SUGGEST_URI_PATH_QUERY, SEARCH);
  uriMatcher.addURI("com.paad.provider.earthquake",
    SearchManager.SUGGEST_URI_PATH_QUERY + "/*", SEARCH);
  uriMatcher.addURI("com.paad.provider.earthquake",
    SearchManager.SUGGEST_URI_PATH_SHORTCUT, SEARCH);
```

```
  uriMatcher.addURI("com.paad.provider.earthquake",
    SearchManager.SUGGEST_URI_PATH_SHORTCUT + "/*", SEARCH);
}
```

6. Измените метод getType таким образом, чтобы он возвращал тип MIME, соответствующий результатам поиска.

```
@Override
public String getType(Uri uri) {
  switch (uriMatcher.match(uri)) {
    case QUAKES : return "vnd.android.cursor.dir/vnd.paad.earthquake";
    case QUAKE_ID: return "vnd.android.cursor.item/vnd.paad.earthquake";
    case SEARCH : return SearchManager.SUGGEST_MIME_TYPE;
    default: throw new IllegalArgumentException("Unsupported URI: " +
uri);
  }
}
```

7. Для внесения окончательных изменений в Источник данных отредактируйте метод query. Он должен вести поисковый запрос и возвращать результат, используя проекцию, созданную на шаге 4. Это позволит Виджету быстрого поиска выводить подсказки при наборе, а ваша Активность сможет отображать поисковые результаты.

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
String[]
                    selectionArgs, String sort) {
  SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
  qb.setTables(EARTHQUAKE_TABLE);

  // Если идет запрос к одной строке, ограничьте результирующий набор
данных этой строкой.
  switch (uriMatcher.match(uri)) {
    case QUAKE_ID: qb.appendWhere(KEY_ID + "=" + uri.getPathSegments().
get(1));
                   break;
    case SEARCH  : qb.appendWhere(KEY_DETAILS + " LIKE \"%" +
                         uri.getPathSegments().get(1) + "%\"");
                   qb.setProjectionMap(SEARCH_PROJECTION_MAP);
                   break;
    default      : break;
  }

  [ ... ранее написанный код ... ]
}
```

8. Создайте новую Активность, необходимую для отображения результатов поиска. Наследуйте класс ListActivity и назовите его EarthquakeSearch. Он будет выводить результат поискового запроса, поэтому вам нужно извлечь этот запрос из Намерения, с помощью которого было запущено приложение, и использовать его для получения результатов из Источника данных EarthquakeProvider. Создайте объект SimpleCursorAdapter,

чтобы привязать результирующий Курсор к элементу ListView, принадлежащему ранее созданной Активности.

```
import android.app.ListActivity;
import android.app.SearchManager;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.widget.SimpleCursorAdapter;

public class EarthquakeSearch extends ListActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    String searchTerm = getIntent().getStringExtra(SearchManager.USER_
QUERY);
    String searchQuery = Uri.withAppendedPath(EarthquakeProvider.SEARCH_
URI,
                                              searchTerm);

    Cursor c = getContentResolver().query(searchQuery, null, null, null,
null);
    startManagingCursor(c);

    String[] from = new String[] {SearchManager.SUGGEST_COLUMN_TEXT_1};
    int[] to = new int[] {android.R.id.text1};
    SimpleCursorAdapter searchResults = new SimpleCursorAdapter(this,
      android.R.layout.simple_list_item_1, c, from, to);
      setListAdapter(searchResults);
  }
}
```

9. Откройте манифест приложения и добавьте в него новую Активность EarthquakeSearch. Не забудьте включить в нее Фильтр намерений для действия SEARCH в категории DEFAULT. Необходимо добавить тег <meta-data>, который ссылается на поисковый ресурс, созданный на шаге 2.

```
<activity android:name=".EarthquakeSearch" android:label="Earthquake
Search">
  <intent-filter>
    <action android:name="android.intent.action.SEARCH" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
  <meta-data
    android:name="android.app.searchable"
    android:resource="@xml/searchable"
  />
</activity>
```

10. В завершение вернитесь к манифесту и добавьте в узел <application> тег <meta-data>, который описывает Активность EarthquakeSearch в качестве поискового источника для приложения по умолчанию.

```
<application android:icon="@drawable/icon">
  <meta-data
    android:name="android.app.default_searchable"
    android:value=".EarthquakeSearch"
  />
  [ ... существующий код в узле application ... ]
</application>
```

Если вы запустите это приложение, нажатие аппаратной кнопки поиска в любой его части приведет к появлению поисковой строки, которая будет выводить подсказки по мере того, как вы набираете свой запрос. Чтобы ваши поисковые результаты стали доступными в Виджете быстрого поиска, необходимо пройти в Settings ▸ Search ▸ Searchable Items и выбрать элемент Earthquake.

Создание Живых обоев

Живые обои — это новый способ, с помощью которого компонент приложения можно добавить на домашний экран. Впервые он был представлен в Android версии 2.1 (API level 7). Живые обои позволяют создавать динамические, интерактивные подложки для домашнего экрана, предлагая новый путь для отображения на нем пользовательской информации.

Живые обои используют объект Surface, чтобы рисовать динамические изображения и отслеживать нажатия на экране. Это дает возможность пользователям взаимодействовать с домашним экраном.

Чтобы создать новые Живые обои, понадобятся три компонента:

- ресурс в формате XML, содержащий описание Живых обоев;
- реализация класса WallpaperService;
- реализация класса Engine (который принадлежит WallpaperService).

Создание ресурса для описания Живых обоев

Ресурс для описания Живых обоев представляет собой файл в формате XML, хранящийся в каталоге res/xml. Используйте атрибуты тега `<wallpaper>`, чтобы определить имя автора, описание и миниатюру для предпросмотра в галерее Живых обоев. Вы также можете задействовать тег `settingsActivity` для указания Активности, которая должна загрузиться при вызове настроек этих обоев.

В листинге 10.25 показан пример ресурса для описания Живых обоев.

Листинг 10.25. Ресурс для описания Живых обоев

```
<wallpaper xmlns:android="http://schemas.android.com/apk/res/android"
  android:author="@string/author"
  android:description="@string/description"
  android:thumbnail="@drawable/wallpapericon"
/>
```

Обратите внимание, что нужно ссылаться на уже существующие строковые ресурсы в атрибутах author и description. Строковые константы не подходят.

Создание Сервиса для Живых обоев

Наследуйте класс WallpaperService — Сервис, который содержит экземпляр класса Engine.

Все операции по отрисовке Живых обоев и взаимодействию с ними содержатся в классе WallpaperService.Editor, который мы рассмотрим ниже в этой главе. Переопределите обработчик onCreateEngine, чтобы возвращать новый экземпляр WallpaperService.Editor, как показывается в листинге 10.26.

Листинг 10.26. Сервис для Живых обоев

```
public class MyWallpaperService extends WallpaperService {
  @Override
  public Engine onCreateEngine() {
    return new MyWallpaperServiceEngine();
  }
}
```

Как только вы создали этот Сервис, добавьте его в манифест вашего приложения с помощью тега <service>. Живые обои должны включать Фильтр намерений, чтобы отслеживать действие android.service.wallpaper.WallpaperService, а также узел <meta-data>, в котором в качестве атрибута name указывается значение android.service.wallpaper, благодаря чему обеспечивается привязка к ресурсу, описанному в предыдущем разделе.

Ваш Сервис также должен иметь полномочие android.permission.BIND_WALLPAPER, которое определяется в атрибуте android.permission. В листинге 10.27 показывается, как добавить Живые обои из листинга 10.26 в манифест приложения.

Листинг 10.27. Добавление WallpaperService в манифест

```
<service android:name=".MyWallpaperService"
  android.permission="android.permission.BIND_WALLPAPER">
  <intent-filter>
    <action android:name="android.service.wallpaper.WallpaperService" />
  </intent-filter>
  <meta-data
    android:name="android.service.wallpaper"
    android:resource="@xml/wallpaper"
  />
</service>
```

Создание движка для Сервиса Живых обоев

В классе WallpaperService.Engine определяются непосредственно сами Живые обои.

Этот класс инкапсулирует работу с объектом Surface, с помощью которого обои выводятся на экран, и обрабатывает события нажатия. Surface — специальный холст для рисования, поддерживающий обновления из фоновых потоков, что делает его идеальным средством для создания плавной, динамической и интерактивной анимации. Объект Surface и обработка нажатий более подробно описаны в главе 15.

Чтобы реализовать собственный движок для Сервиса Живых обоев, наследуйте класс WallpaperService.Engine. Прежде чем начать рисовать с помощью объекта Surface, нужно дождаться его полной инициализации, используя обработчик onSurfaceCreated.

Класс Engine включает в себя обработчик onTouchEvent и обработчик onOffsetsChanged: первый обеспечивает взаимодействие с пользователем, второй срабатывает в момент, когда сдвигается родительская Активность (как правило, это домашний экран).

В листинге 10.28 показан каркас для реализации наследника класса WallpaperService.Engine. Откройте главу 15, чтобы получить более детальную информацию, как рисовать с помощью объекта Surface, а также как использовать обработчик onTouchEvent и события движения.

Листинг 10.28. Каркас движка для Сервиса Живых обоев

```
public class MyWallpaperServiceEngine extends WallpaperService.Engine {
  @Override
  public void onCreate(SurfaceHolder surfaceHolder) {
    super.onCreate(surfaceHolder);
    // TODO Обрабатывать инициализацию.
  }

  @Override
  public void onOffsetsChanged(float xOffset, float yOffset,
                               float xOffsetStep, float yOffsetStep,
                               int xPixelOffset, int yPixelOffset) {
    super.onOffsetsChanged(xOffset, yOffset, xOffsetStep, yOffsetStep,
                           xPixelOffset, yPixelOffset);
    // TODO Обрабатывать события сдвига домашнего экрана.
  }
  @Override
  public void onTouchEvent(MotionEvent event) {
    super.onTouchEvent(event);
    // TODO Обрабатывать события нажатия и движения.
  }

  @Override
  public void onSurfaceCreated(SurfaceHolder holder) {
    super.onSurfaceCreated(holder);
    // TODO Объект Surface был создан, теперь нужно запустить поток,
    // в котором будут рисоваться обои.
  }
}
```

Резюме

В этой главе вы научились создавать виджеты и Живые каталоги для своего приложения.

В частности, вы узнали:

- как реализовывать виджеты и добавлять их в свои приложения;
- контролировать частоту обновлений своего виджета, передавая минимальное значение для этой частоты или используя Намерения и механизм Сигнализации;
- обновлять пользовательский интерфейс своего виджета с помощью объекта RemoteViews;
- добавлять элементы интерактивности в виджет;
- создавать и регистрировать Живые каталоги для Источника данных приложения;
- добавлять проекцию для своего Источника данных, делая его совместимым с Живым каталогом;
- создавать и использовать Живые обои;
- добавлять поисковые возможности в свое приложение и выводить результаты в Виджете быстрого поиска.

В следующей главе вы изучите аудиовизуальные API, доступные в Android, познакомитесь с возможностями проигрывания и записи мультимедийных файлов с помощью микрофона и камеры.

Глава 11
АУДИО, ВИДЕО И КАМЕРА

Содержание главы

- Воспроизведение аудио- и видеоданных с помощью Медиапроигрывателя (Media Player).
- Упаковка звуковых файлов в качестве ресурсов приложения.
- Использование объекта VideoView для проигрывания видео.
- Запись аудио и видео с помощью объекта MediaRecorder.
- Запись видео и создание снимков с помощью Намерений.
- Предварительный просмотр записываемого видео и отображение видеопотоков с камеры в режиме реального времени.
- Создание снимков и управление камерой.
- Чтение и редактирование данных EXIF, содержащихся в изображении.
- Добавление мультимедийных файлов в MediaStore.
- Операции с необработанными аудиоданными.
- Использование возможностей по распознаванию речи.

Единственными современными устройствами, которые могут конкурировать с мобильными телефонами с точки зрения популярности, можно назвать портативные цифровые плееры. В связи с этим мультимедийные возможности мобильных устройств имеют большое значение для многих потребителей.

Android — открытая платформа, в основе которой принцип независимости от конкретного поставщика данных. Благодаря этому Android предоставляет мультимедийные API, совместимые с проигрыванием и записью широкого диапазона всевозможных форматов изображений, аудио и видео — как локально, так и в потоковом режиме.

API для камеры наряду с платформой OpenCORE обеспечивают полнофункциональные мультимедийные возможности и позволяют использовать их внутри собственных приложений.

В этой главе вы узнаете, как проигрывать и записывать мультимедийные данные, включая аудио, видео и статические изображения, как использовать камеру, чтобы создавать снимки и запускать предварительный просмотр, а также как записать живое видео.

Вы также научитесь работать с несжатыми аудиоданными с помощью классов AudioTrack и AudioRecord, добавлять только что записанные медиафайлы в MediaStore, использовать возможности по распознаванию речи, дополняя свои приложения поддержкой голосового ввода.

Проигрывание аудио и видео

Android содержит полноценный Медиапроигрыватель, который упрощает проигрывание аудио- и видеоданных. В этом разделе рассматривается использование данного объекта для управления воспроизведением мультимедийной информации внутри ваших приложений.

В Android 2.1 (API level 7) поддержка проигрывания следующих мультимедийных форматов — часть базового фреймворка (имейте в виду, что некоторые устройства могут поддерживать проигрывание дополнительных файловых форматов):

- аудио:
  - AAC LC/LTP;
  - HE-AACv1 (AAC+);
  - HE-AACv2 (Enhanced AAC+);
  - AMR-NB AMR-WB;
  - MP3;
  - MIDI;
  - Ogg Vorbis;
  - PCM / WAVE;
- видео:
  - H.263;
  - H.264 AVC;
  - MPEG-4 SP.

Знакомство с Медиапроигрывателем

За проигрывание мультимедийных файлов в Android отвечает класс MediaPlayer. Вы можете воспроизводить медиаданные, размещенные в ресурсах приложения, локальных файлах, Источниках данных или в сетевом

потоке. В каждом случае вы как разработчик абстрагируетесь от форматов файлов и типов мультимедийных данных.

В объекте MediaPlayer управление аудио- и видеофайлами реализовано в виде машины состояний. Если говорить более простым языком, режимы, через которые проходит эта машина состояний, можно описать так:

- инициализация Медиапроигрывателя с помощью заданных мультимедийных данных;
- подготовка объекта MediaPlayer к воспроизведению;
- запуск воспроизведения;
- временная или полная остановка во время воспроизведения;
- завершение воспроизведения.

Более детальное и исчерпывающее описание машины состояний Медиапроигрывателя вы можете найти на сайте для разработчиков по адресу http://developer.android.com/reference/android/media/MediaPlayer.html#StateDiagram.

Чтобы проигрывать мультимедийный ресурс, необходимо создать новый экземпляр класса MediaPlayer, инициализировать его с помощью источника медиаданных и подготовить к воспроизведению.

В следующем разделе рассказывается, как нужно инициализировать и подготавливать объект MediaPlayer. После этого вы научитесь управлять процессом воспроизведения, запуская, останавливая и ставя его на паузу, а также перемещаясь по временной шкале.

В любом случае, при завершении процесса проигрывания необходимо вызвать метод release из объекта MediaPlayer, чтобы освободить соответствующие ресурсы:

```
mediaPlayer.release();
```

Android поддерживает ограниченное число одновременно работающих объектов MediaPlayer. Не освобождая их, вы рискуете столкнуться с выбросом исключения, когда система исчерпает ресурсы.

Подготовка аудиоданных к воспроизведению

Есть немало способов, с помощью которых можно воспроизводить аудиоданные в Медиапроигрывателе. Вы можете включать эти данные в виде ресурса приложения, проигрывать их из локальных файлов, Источников данных или сетевых потоков через удаленный адрес URL.

Упаковка аудиофайла в виде ресурса приложения

Вы можете включать аудиофайлы в дистрибутив своего приложения — в каталог res/raw иерархии ресурсов.

Во время упаковки в приложение необработанные ресурсы не сжимаются и никак не изменяются. В связи с этим данный способ идеален для хранения предварительно сжатых данных, таких как звуковые файлы.

Чтобы получить доступ к необработанному ресурсу, используйте имя файла в нижнем регистре без расширения, как показано в листинге 11.1.

Инициализация аудиоданных для воспроизведения

Чтобы начать воспроизведение аудиоданных с помощью Медиапроигрывателя, необходимо создать объект MediaPlayer и назначить для него источник с мультимедийной информацией (в нашем случае с аудио).

Для воспроизведения аудио с помощью Медиапроигрывателя вы можете использовать статический метод create, передавая ему в качестве параметров Контекст приложения, а также одно из следующих значений (как показано в листинге 11.1):

- идентификатор ресурса;
- путь URI к локальному файлу (используя схему `file://`);
- путь URI к удаленному ресурсу с аудио, предоставленному в виде URL;
- путь URI к записи внутри локального Источника данных.

Обратите внимание, что для объекта MediaPlayer, возвращаемого методом create, уже был вызван метод prepare. Важно, чтобы вы не вызвали его повторно.

Листинг 11.1. Инициализация аудиоданных для воспроизведения

```
Context appContext = getApplicationContext();

MediaPlayer resourcePlayer = MediaPlayer.create(appContext,
  R.raw.my_audio);
MediaPlayer filePlayer = MediaPlayer.create(appContext,
  Uri.parse("file:///sdcard/localfile.mp3"));
MediaPlayer urlPlayer = MediaPlayer.create(appContext,
  Uri.parse("http://site.com/audio/audio.mp3"));
MediaPlayer contentPlayer = MediaPlayer.create(appContext,
  Settings.System.DEFAULT_RINGTONE_URI);
```

В качестве альтернативного можно использовать метод setDataSource из уже созданного экземпляра MediaPlayer. В виде единственного параметра этот метод может принять путь к файлу, путь URI к Источнику данных, адрес URL мультимедийного потока или файловый дескриптор.

Используя данный подход, важно не забыть вызвать метод prepare из объекта MediaPlayer, прежде чем начинать воспроизведение, как показано в листинге 11.2.

Листинг 11.2. Использование методов setDataSource и prepare для инициализации воспроизведения аудио

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource("/sdcard/test.3gp");
mediaPlayer.prepare();
```

ПРИМЕЧАНИЕ

Если вы передаете адрес URL, который ссылается на файл в Интернете, удаленный сервер должен поддерживать последовательную загрузку с помощью протоколов RTSP или HTTP.

Подготовка видеоданных к воспроизведению

Воспроизведение видеоданных немного сложнее, чем аудио. Чтобы отобразить видео, необходимо указать поверхность, на которой оно будет выводиться. В следующих разделах описываются два способа воспроизведения видеоданных.

Первый предусматривает использование элемента VideoView и инкапсулирует создание и размещение поверхности для отображения видео, а также его подготовку внутри Медиапроигрывателя.

Второй позволяет указать собственную поверхность для воспроизведения, управляя исходным экземпляром MediaPlayer напрямую.

Воспроизведение видео с помощью элемента VideoView

Самый простой способ проигрывания видео — использование элемента VideoView. Он содержит поверхность (объект Surface), на которую выводится картинка, а также инкапсулирует все операции по управлению Медиапроигрывателем.

Как и MediaPlayer, VideoView поддерживает воспроизведение видео из локальных источников или из потока.

Данный элемент скрывает от разработчика инициализацию Медиапроигрывателя, предоставляя удобный API. Чтобы задать видео для воспроизведения, вызовите метод setVideoPath или setVideoUri. В качестве единственного параметра они принимают путь к локальному файлу, путь URI к Источнику данных или адрес удаленного видеопотока:

```
streamingVideoView.setVideoUri("http://www.mysite.com/videos/
myvideo.3gp");
localVideoView.setVideoPath("/sdcard/test2.3gp");
```

Завершив инициализацию, вы получаете возможность управлять воспроизведением с помощью методов start, stopPlayback, pause и seekTo. VideoView также включает метод setKeepScreenOn для предотвращения отключения подсветки экрана во время проигрывания.

В листинге 11.3 показан простой каркас, содержащий инициализацию VideoView и управление воспроизведением видео.

Листинг 11.3. Воспроизведение видео с помощью элемента VideoView

```
VideoView videoView = (VideoView)findViewById(R.id.surface);
videoView.setKeepScreenOn(true);
videoView.setVideoPath("/sdcard/test2.3gp");
if (videoView.canSeekForward())
  videoView.seekTo(videoView.getDuration()/2);
videoView.start();
[ . . . выполняем какие-либо действия . . . ]
videoView.stopPlayback();
```

Подготовка поверхности для воспроизведения видеоданных

Первым делом при использовании Медиапроигрывателя с целью показа видео необходимо подготовить поверхность, на которой это видео будет отображаться. Для этого Медиапроигрывателю с помощью метода setDisplay нужно передать объект класса SurfaceHolder.

ПРИМЕЧАНИЕ

Если вы не передадите SurfaceHolder для объекта MediaPlayer, компонент с видео не будет отображен.

Чтобы добавить объект SurfaceHolder в разметку вашего пользовательского интерфейса, используйте элемент SurfaceView, как показано в листинге 11.4.

Листинг 11.4. Разметка с элементом SurfaceView

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <SurfaceView
    android:id="@+id/surface"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center">
  </SurfaceView>
</LinearLayout>
```

SurfaceView — обертка вокруг класса SurfaceHolder, который в свою очередь служит оберткой класса Surface, используемого для обновления изображения из фоновых потоков.

Более детально элемент SurfaceView рассмотрен в главе 15, а в листинге 11.5 показан каркас для инициализации SurfaceView внутри Активности, которая передается в объект MediaPlayer.

Обратите внимание, что при этом вы должны реализовать интерфейс SurfaceHolder.Callback. Объект SurfaceHolder создается в асинхронном режиме, поэтому прежде чем передавать его в MediaPlayer, нужно дождаться, когда сработает обработчик surfaceCreated.

Листинг 11.5. Инициализация и передача объекта SurfaceView в Медиапроигрыватель

```
public class MyActivity extends Activity implements SurfaceHolder.
Callback
{
  private MediaPlayer mediaPlayer;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mediaPlayer = new MediaPlayer();

    SurfaceView surface = (SurfaceView)findViewById(R.id.surface);
    SurfaceHolder holder = surface.getHolder();
    holder.addCallback(this);
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    holder.setFixedSize(400, 300);
  }

  public void surfaceCreated(SurfaceHolder holder) {
    try {
      mediaPlayer.setDisplay(holder);
    } catch (IllegalArgumentException e) {
      Log.d("MEDIA_PLAYER", e.getMessage());
    } catch (IllegalStateException e) {
      Log.d("MEDIA_PLAYER", e.getMessage());
    } catch (IOException e) {
      Log.d("MEDIA_PLAYER", e.getMessage());
    }
  }

  public void surfaceDestroyed(SurfaceHolder holder) {
    mediaPlayer.release();
  }

  public void surfaceChanged(SurfaceHolder holder,
                             int format, int width, int height) { }
}
```

Инициализация видеоданных для воспроизведения

Успешно создав и передав объект SurfaceHolder в Медиапроигрыватель, используйте метод setDataSource, чтобы указать локальный адрес, URL или путь URI Источника данных, указывающие на ресурс с видео, которое нужно воспроизвести.

Как и в случае с проигрыванием аудиоданных, если вы указываете адрес URL к удаленному файлу, сервер, на котором он находится, должен поддерживать последовательную загрузку с помощью протоколов RTSP или HTTP.

Выбрав источник с медиаданными, вызовите метод prepare, чтобы инициализировать объект MediaPlayer и подготовить его для воспроизведения, как показано в листинге 11.6.

Листинг 11.6. Инициализация видеоданных для воспроизведения с помощью объекта MediaPlayer

```
public void surfaceCreated(SurfaceHolder holder) {
  try {
    mediaPlayer.setDisplay(holder);
    mediaPlayer.setDataSource("/sdcard/test2.3gp");
    mediaPlayer.prepare();
    mediaPlayer.start();
  } catch (IllegalArgumentException e) {
    Log.d("MEDIA_PLAYER", e.getMessage());
  } catch (IllegalStateException e) {
    Log.d("MEDIA_PLAYER", e.getMessage());
  } catch (IOException e) {
    Log.d("MEDIA_PLAYER", e.getMessage());
  }
}
```

ВНИМАНИЕ

В отличие от ресурсов с аудиоданными, Android не поддерживает проигрывание видео, включенного в проект в качестве ресурса. Вы не можете также использовать статический метод create для упрощенного создания объекта MediaPlayer и путь URI, ссылающийся на локальный файл с помощью схемы file://.

Управление воспроизведением

Подготовив Медиапроигрыватель, вызовите метод start, чтобы начать воспроизведение соответствующих мультимедийных данных:

```
mediaPlayer.start();
```

Используйте методы stop и pause, чтобы остановить или приостановить проигрывание.

Объект MediaPlayer также предоставляет методы getDuration (позволяет получить длину воспроизводимого аудио или видео) и getCurrentPosition (возвращает текущую позицию). Чтобы перейти к определенной позиции, воспользуйтесь методом seekTo, как показано в листинге 11.7.

Листинг 11.7. Управление воспроизведением

```
mediaPlayer.start();

int pos = mediaPlayer.getCurrentPosition();
int duration = mediaPlayer.getDuration();

mediaPlayer.seekTo(pos + (duration-pos)/10);

[ ... ждем перехода к позиции . . . ]

mediaPlayer.stop();
```

Управление мультимедийным выводом при воспроизведении

Медиапроигрыватель предоставляет управление громкостью, запрет отключения подсветки экрана во время воспроизведения, а также возможность проигрывать файлы в режиме повторения.

В настоящее время нельзя воспроизводить аудиоданные одновременно с разговором по телефону. Медиапроигрыватель всегда использует стандартное устройство для вывода звука — динамик или подключенную по Bluetooth гарнитуру.

Используйте методы isLooping и setLooping, чтобы определить, будет ли файл воспроизводиться в режиме повторения.

```
if (!mediaPlayer.isLooping())
  mediaPlayer.setLooping(true);
```

Чтобы подсветка экрана автоматически не отключалась при воспроизведении, применяйте метод setScreenOnWhilePlaying. Он предпочтительнее, чем установка запрета на отключение вручную, поскольку не требует дополнительных полномочий. Этот механизм более подробно описан в главе 15.

```
mediaPlayer.setScreenOnWhilePlaying(true);
```

Во время воспроизведения с помощью метода setVolume можно управлять громкостью каждого аудиоканала. В качестве параметров этот метод принимает скалярные значения с плавающей точкой в диапазоне между 0 и 1 для обоих каналов (где 0 — полная тишина, а 1 — максимальная громкость).

```
mediaPlayer.setVolume(1f, 0.5f);
```

ПРИМЕЧАНИЕ

При воспроизведении видеоресурсов вы можете использовать метод getFrame, чтобы получить заданный кадр.

Запись аудио- и видеоданных

Android предлагает два разных пути для записи аудио и видео внутри приложения.

Самый простой способ — использование Намерений для запуска стандартного приложения, управляющего камерой. Это позволяет указать качество видео и место, куда его сохранять. Вся работа по записи видео, взаимодействию с пользователем и отлавливанию ошибок «ложится на плечи» стандартного приложения.

Если нужен более тонкий контроль над пользовательским интерфейсом или параметрами записи и вы хотите заменить стандартное приложение, можете воспользоваться классом MediaRecorder.

Использование Намерений в сочетании с объектом MediaRecorder

Самый простой способ инициировать запись видео — использование статической константы ACTION_VIDEO_CAPTURE из объекта MediaStore в качестве параметра для Намерения, которое, в свою очередь, передается методу startActivityForResult.

```
startActivityForResult(new Intent(MediaStore.ACTION_VIDEO_CAPTURE),
                                  RECORD_VIDEO);
```

Этот код запускает стандартную управляющую видеокамерой Активность, позволяя пользователям начать, остановить, просмотреть и повторить запись видео, что освобождает вас от необходимости создавать свое собственное приложение для этих нужд.

Для записи видео поддерживаются два необязательных параметра, доступных в качестве статических констант класса MediaStore.

- EXTRA_OUTPUT. По умолчанию записанное видео сохраняется в источнике MediaStore. Если хотите сохранить его в другом месте, используйте этот параметр для указания альтернативного пути URI.
- EXTRA_VIDEO_QUALITY. В процессе записи можно указать качество картинки с помощью целочисленного значения. В настоящее время доступны два варианта: 0 — для низкого качества (подходит для MMS) и 1 — для видео с высоким разрешением. По умолчанию выбирается второй вариант.

В листинге 11.8 показано, как с помощью описанных выше действий записать новый видеофайл с высоким качеством, сохраняя его по указанному пути URI или в стандартный источник MediaStore.

Листинг 11.8. Запись видео с помощью Намерения

```
private static int RECORD_VIDEO = 1;
private static int HIGH_VIDEO_QUALITY = 1;
private static int MMS_VIDEO_QUALITY = 0;

private void recordVideo(Uri outputpath) {
  Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

  if (outputpath != null)
    intent.putExtra(MediaStore.EXTRA_OUTPUT, output);
  intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, HIGH_VIDEO_QUALITY);

  startActivityForResult(intent, RECORD_VIDEO);
}

@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
  if (requestCode == RECORD_VIDEO) {
    Uri recordedVideo = data.getData();
    // TODO Сделать что-нибудь с записанным видеофайлом
  }
}
```

Использование объекта MediaRecorder

Запись мультимедийных файлов — прерогатива класса MediaRecorder. С его помощью можно записывать аудио- и видеоданные, которые потом могут быть использованы вашим приложением или сохранены в источник MediaStore.

Сперва необходимо создать объект MediaRecorder.

```
MediaRecorder mediaRecorder = new MediaRecorder();
```

Прежде чем записывать мультимедийные файлы в Android, приложение должно получить полномочия RECORD_AUDIO и/или RECORD_VIDEO. Добавьте в манифест приложения тег `uses-permission` для каждого из этих полномочий.

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.RECORD_VIDEO"/>
```

MediaRecorder позволяет указать источник для аудио и видео, формат итогового файла, а также кодировщики, использующиеся для записи.

Как и MediaPlayer, объект MediaRecorder управляет записью с помощью машины состояний. Это значит, что порядок, в котором вы настраиваете запись и управляете ею, играет важную роль.

Иными словами, режимы, через которые проходит эта машина состояний, можно описать так:

1) создание нового объекта MediaRecorder;
2) назначение источника записи;
3) определение итогового формата;
4) указание кодировщиков для аудио и видео, частоту кадров и результирующий размер;
5) выбор файла, в который будет производиться запись;
6) подготовка к записи;
7) запись;
8) конец записи.

Более детальное и исчерпывающее описание машины состояний объекта MediaRecorder можно найти на сайте для разработчиков под Android по адресу http://developer.android.com/reference/android/media/MediaRecorder.html.

Закончив запись мультимедийного файла, вызовите метод release из объекта MediaRecorder, чтобы освободить соответствующие ресурсы.

```
mediaRecorder.release();
```

Запись видео: настройка и управление

Как следует из модели состояний, описанной выше, прежде чем начинать запись, необходимо указать входящий источник, итоговый формат, кодировщики для аудио и видео, а также файл, в который будет вестить запись, — именно в таком порядке.

Методы setAudioSource и setVideoSource предлагают обозначить одну из статических констант, хранящихся в MediaRecorder.AudioSource и MediaRecorder.VideoSource, которые определяют источники для аудио- и видеоданных соответственно.

Задав источники для записи, выберите итоговый формат с помощью метода setOutputFormat, передав ему одну из констант, хранящихся в MediaRecorder.OutputFormat.

Укажите кодировщики для аудио и видео с помощью методов set[audio/video]Encoder, используя константы из класса MediaRecorder.[Audio/Video]Encoder. При желании установите частоту кадров и размер итогового видео.

В завершение с помощью метода setOutputFile передайте файл, в который будет производиться запись. И только после этого вызовите метод prepare.

В листинге 11.9 показано, как настроить MediaRecorder для записи аудио- и видеоданных с микрофона и камеры соответственно, используя итоговый формат и кодировщики по умолчанию и сохраняя запись на карту SD.

Листинг 11.9. Настройка объекта MediaRecorder

```
MediaRecorder mediaRecorder = new MediaRecorder();

// Настройте источники для записи
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);

// Установите итоговый формат
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);

// Укажите кодировщики для аудио и видео
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);

// Задайте файл для сохранения записи
mediaRecorder.setOutputFile("/sdcard/myoutputfile.mp4");

// Подготовьте MediaRecorder к записи
mediaRecorder.prepare();
```

Чтобы начать запись, вызовите метод start, как показано в дополнении к листингу 11.9.

```
mediaRecorder.start();
```

ВНИМАНИЕ

Метод setOutputFile нужно вызвать перед методом prepare и после метода setOutputFormat, иначе произойдет выброс исключения IllegalStateException.

Когда закончите, вызовите метод stop, чтобы завершить запись, и release, чтобы освободить ресурсы объекта MediaRecorder.

```
mediaRecorder.stop();
mediaRecorder.release();
```

Предварительный просмотр записываемого видео

В процессе записи видео, как правило, не помешает отображать записываемую картинку в режиме реального времени. Используя метод setPreviewDisplay, вы можете задать объект Surface для показа видеопотока во время записи.

Работает данный подход так же, как воспроизведение видео с помощью Медиапроигрывателя, описанное ранее в этой главе.

Начните с создания новой Активности, которая включает элемент пользовательского интерфейса SurfaceView и реализует интерфейс SurfaceHolder. Callback.

Создав объект SurfaceHolder, передайте его в метод setPreviewDisplay, принадлежащий MediaRecorder, как показано в листинге 11.10.

Как только вы вызовете метод prepare, на экране в режиме реального времени начнет отображаться записываемый видеопоток.

Листинг 11.10. Предварительный просмотр записываемого видео

```
public class MyActivity extends Activity implements SurfaceHolder.
Callback
{
  private MediaRecorder mediaRecorder;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    SurfaceView surface = (SurfaceView)findViewById(R.id.surface);
    SurfaceHolder holder = surface.getHolder();
    holder.addCallback(this);
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    holder.setFixedSize(400, 300);
  }

  public void surfaceCreated(SurfaceHolder holder) {
    if (mediaRecorder == null) {
      try {
        mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
        mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);

        mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.
DEFAULT);

        mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.
DEFAULT);
        mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.
DEFAULT);
        mediaRecorder.setOutputFile("/sdcard/myoutputfile.mp4");

        mediaRecorder.setPreviewDisplay(holder.getSurface());
        mediaRecorder.prepare();
      } catch (IllegalArgumentException e) {
        Log.d("MEDIA_PLAYER", e.getMessage());
      } catch (IllegalStateException e) {
        Log.d("MEDIA_PLAYER", e.getMessage());
      } catch (IOException e) {
        Log.d("MEDIA_PLAYER", e.getMessage());
      }
```

```
    }
  }

  public void surfaceDestroyed(SurfaceHolder holder) {
    mediaRecorder.release();
  }

  public void surfaceChanged(SurfaceHolder holder,
                             int format, int width, int height) { }
}
```

Использование камеры и создание снимков

Вместе с ростом популярности цифровых камер (особенно внутри мобильных телефонов), цены на них значительно снизились, а сами камеры стали намного компактнее. Сейчас трудно найти мобильный телефон без камеры, и устройства на базе Android не исключение.

Модель G1 выпущена в 2008 г., оснащалась она камерой в 3,2 мегапиксела. На сегодняшний день некоторые устройства оснащены 5-мегапиксельными камерами, а одна модель — сенсором с разрешением 8,1 мегапиксела.

В следующих разделах описан механизм, который управляет камерой и делает снимки внутри приложения.

Использование Намерений для создания снимков

Самый простой способ сделать снимок с помощью камеры — использовать статическую константу ACTION_IMAGE_CAPTURE из объекта MediaStore для создания Намерения, которое потом нужно передать методу startActivityForResult.

```
startActivityForResult(new Intent(MediaStore.ACTION_IMAGE_CAPTURE),
                       TAKE_PICTURE);
```

Данный код запустит Активность для управления камерой, позволяя таким образом пользователям вручную изменять настройки изображения (это освобождает вас от необходимости создавать свое собственное приложение для этих нужд).

Процесс создания снимков предусматривает два режима.

- **Миниатюра.** По умолчанию фотография возвращается в виде объекта Bitmap, содержащего миниатюру. Этот объект находится в параметре data, передаваемом в метод onActivityResult. Как показано в листинге 11.11, чтобы получить миниатюру в виде объекта Bitmap, нужно вызвать метод getParcelableExtra из Намерения, передав ему строковое значение data.

- **Полноценное изображение.** Если вы укажете исходящий путь URI с помощью параметра MediaStore.EXTRA_OUTPUT в запущенном Намерении, полноразмерное изображение, снятое камерой, сохранится в заданном месте. В таком случае в метод onActivityResult не будет передана миниатюра, а итоговое Намерение продемонстрирует значение null.

В листинге 11.11 показано, как при создании снимка получать миниатюру или полноценное изображение, используя Намерение.

Листинг 11.11. Создание снимка с помощью Намерения

```
private static int TAKE_PICTURE = 1;
private Uri outputFileUri;

private void getThumbailPicture() {
  Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
  startActivityForResult(intent, TAKE_PICTURE);
}

private void saveFullImage() {
  Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
  File file = new File(Environment.getExternalStorageDirectory(),
                       "test.jpg");
  outputFileUri = Uri.fromFile(file);
  intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
  startActivityForResult(intent, TAKE_PICTURE);
}

@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
  if (requestCode == TAKE_PICTURE) {
    Uri imageUri = null;

    // Check if the result includes a thumbnail Bitmap
    if (data != null) {
      if (data.hasExtra("data")) {
      Bitmap thumbnail = data.getParcelableExtra("data");
      // TODO Какие-то действия с миниатюрой
      }
    }
    else {
      // TODO Какие-то действия с полноценным изображением,
      // сохраненным по адресу outputFileUri
    }
  }
}
```

Сделав снимок, можете добавить его в источник MediaStore, как показано далее в этой главе, или обработать его, чтобы использовать внутри своего приложения, прежде чем удалить.

Управление камерой и создание снимков

Чтобы получить прямой доступ к аппаратным возможностям камеры, необходимо добавить полномочие CAMERA в манифест своего приложения.

```
<uses-permission android:name="android.permission.CAMERA"/>
```

Используйте класс Camera, чтобы откорректировать настройки камеры, указать параметры изображения и сделать снимок.

Чтобы получить доступ к Сервису камеры, применяйте статический метод open из класса Camera. Когда приложение закончило работу с этим Сервисом, не забудьте освободить его ресурсы с помощью метода release, как показано в листинге 11.12.

Листинг 11.12. Использование камеры

```
Camera camera = Camera.open();
[ . . . Какие-либо операции с камерой . . . ]
camera.release();
```

ПРИМЕЧАНИЕ

Метод Camera.open включает и инициализирует камеру. После его вызова вы можете редактировать настройки, подготавливать поверхность для предварительного просмотра и делать снимки. Эти операции рассмотрены в следующих разделах.

Отслеживание и изменение настроек камеры и параметров изображения

Настройки камеры хранятся в объекте Camera.Parameters, доступ к которому можно получить с помощью метода getParameters, вызванного из экземпляра класса Camera.

Чтобы изменить эти настройки (листинг 11.13), используйте методы set* из объекта Parameters, после чего вызовите метод setParameters, передав ему модифицированный объект.

Листинг 11.13. Чтение и изменение настроек камеры

```
Camera.Parameters parameters = camera.getParameters();
[ . . . внесение изменений . . . ]
camera.setParameters(parameters);
```

В версии Android 2.0 (API level 5) представлены параметры камеры, у каждого из которых свой геттер и сеттер.

- [get/set]SceneMode. Принимает или возвращает статическую строковую константу SCENE_MODE_* из класса Parameters. Каждый

режим описывает определенный тип обстановки («вечеринка», «пляж», «закат» и т. д.).

- [get/set]FlashMode. Принимает или возвращает статическую строковую константу FLASH_MODE_*. Позволяет указать режим освещения (включен, выключен или уменьшение эффекта красных глаз) или режим вспышки.
- [get/set]WhiteBalance. Принимает или возвращает статическую строковую константу WHITE_BALANCE_*, с помощью которой описывается баланс белого цвета для фотографируемой сцены.
- [get/set]ColorEffect. Принимает или возвращает статическую строковую константу EFFECT_*, чтобы изменить способ представления изображения. Вам доступны цветовые эффекты, включая тон сепии или оттенки серого.
- [get/set]FocusMode. Принимает или возвращает статическую строковую константу FOCUS_MODE_*, которая задает режим автоматической фокусировки камеры.

ПРИМЕЧАНИЕ

Большинство описанных выше параметров пригодятся прежде всего в том случае, если вы хотите заменить стандартное приложение для работы с камерой. Тем не менее их также можно задействовать при настройке предварительного просмотра, что позволяет изменять живой видеопоток, создавая приложения дополненной реальности.

Параметрами камеры также можно воспользоваться при чтении и записи настроек размера, качества и формата изображения, миниатюры и картинки, отображаемой для предварительного просмотра.

- **Качество JPEG и миниатюры.** Используйте методы setJpegQuality и setJpegThumbnailQuality, передавая им целочисленные значения от 0 до 100, где 100 — самое высокое качество.
- **Размер изображения, картинки для предварительного просмотра, а также миниатюры.** Применяйте методы setPictureSize, setPreviewSize и setJpegThumbnailSize, чтобы задать высоту и ширину изображению, картинке для предварительного просмотра и миниатюре соответственно.
- **Растровый формат для изображения и картинки при предварительном просмотре.** Используйте методы setPictureFormat и setPreviewFormat, чтобы задать формат изображения, задействуя статическую константу из класса PixelFormat.
- **Частота кадров при предварительном просмотре.** Применяйте метод setPreviewFrameRate, чтобы указать частоту кадров для предварительного просмотра в FPS (количество кадров в секунду).

Каждое конкретное устройство потенциально содержит поддержку собственного подмножества этих значений. Класс Camera.Parameters также содержит методы getSupported*, с помощью которых можно найти доступные параметры, чтобы потом показать их пользователю. Перед присвоением значения нужно удостовериться, что данный параметр поддерживается. Подобный подход описан в листинге 11.14.

Проверка параметров на допустимость имеет большое значение при выборе доступного режима для предварительного просмотра или размера изображения, так как камеры в различных устройствах могут обладать разными характеристиками.

Листинг 11.14. Проверка поддержки камерой определенных параметров

```
Camera.Parameters parameters = camera.getParameters();
List<String> colorEffects = parameters.getSupportedColorEffects();
if (colorEffects.contains(Camera.Parameters.EFFECT_SEPIA))
  parameters.setColorEffect(Camera.Parameters.EFFECT_SEPIA);
camera.setParameters(parameters);
```

Отслеживание состояния автоматической фокусировки

Если камера в устройстве поддерживает автоматическую фокусировку (автофокус), вы можете проследить, прошла ли данная операция успешно, используя функцию обратного вызова AutoFocusCallback внутри объекта Camera.

В листинге 11.15 показано, как создать и назначить AutoFocusCallback для объекта Camera. При изменении состояния автофокуса обработчик события onAutoFocus получает в качестве параметров объект Camera и булево значение success, которое сигнализирует об успешности автоматической фокусировки.

Листинг 11.15. Отслеживание состояния автофокуса

```
camera.autoFocus(new AutoFocusCallback() {
  public void onAutoFocus(boolean success, Camera camera) {
    // TODO Использовать значение success
  }
});
```

Использование предварительного просмотра для камеры

Доступ к видеопотоку камеры позволяет добавлять в приложения возможность предварительного просмотра.

Некоторые наиболее впечатляющие приложения для Android используют эту функциональность как основу для реализации дополненной реальности (наложения динамических контекстных данных, например информации о достопримечательностях или различных заведениях поверх живого видеопотока с камеры).

Как и в случае с классами MediaPlayer и MediaRecorde, предварительный просмотр возможен с помощью объекта SurfaceHolder. Чтобы отобразить живой видеопоток с камеры в своем приложении, нужно добавить к пользовательскому интерфейсу элемент SurfaceView. Прежде чем передавать SurfaceHolder в метод setPreviewDisplay из объекта Camera, реализуйте абстрактный интерфейс SurfaceHolder.Callback, чтобы следить за созданием рабочей поверхности.

Вызов метода startPreview инициирует отображения потока, а stopPreview его остановит, как показано в листинге 11.16.

Листинг 11.16. Предварительный просмотр видеопотока с камеры в режиме реального времени

```
public class MyActivity extends Activity implements SurfaceHolder.
Callback {
  private Camera camera;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    SurfaceView surface = (SurfaceView)findViewById(R.id.surface);
    SurfaceHolder holder = surface.getHolder();
    holder.addCallback(this);
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    holder.setFixedSize(400, 300);
  }

  public void surfaceCreated(SurfaceHolder holder) {
    if (mediaRecorder == null) {
      try {
        camera = camera.open();
        camera.setPreviewDisplay(holder);
        camera.startPreview();
        [ . . . Вывод изображения на поверхности . . . ]
      } catch (IOException e) {
        Log.d("CAMERA", e.getMessage());
      }
    }
  }

  public void surfaceDestroyed(SurfaceHolder holder) {
    camera.stopPreview();
    camera.release();
  }
}
```

В главе 15 вы узнаете больше о поверхностях для отображения видеоданных. Обратите внимание на пример, входящий в состав Android SDK, в котором описывается использование объекта SurfaceView для отображения предварительного просмотра видеопотока с камеры в режиме реального времени.

Вы также можете назначить PreviewCallback для срабатывания на каждом новом кадре — это позволит манипулировать с картинкой или отображать каждый кадр отдельно.

Вызовите метод setPreviewCallback из объекта Camera, передав ему в качестве параметра новую реализацию PreviewCallback и переопределив обработчик onPreviewFrame, как показано в листинге 11.17.

Листинг 11.17. Назначение функции обратного вызова для предварительного просмотра

```
camera.setPreviewCallback(new PreviewCallback() {
  public void onPreviewFrame(byte[] _data, Camera _camera) {
    // TODO Какие-то действия с изображением из предварительного
    // просмотра.
  }
});
```

Обработчик onPreviewFrame получит каждый кадр в виде массива байтов.

Создание фотографий

Сделайте снимок, вызвав метод takePicture из объекта Camera, передав ему в качестве параметров ShutterCallback и две реализации PictureCallback (одну для необработанных изображений, другую для сжатых в формат JPEG).

Каждая из этих двух функций обратного вызова получит массив байтов — изображение в соответствующем формате, тогда как ShutterCallback сработает сразу после закрытия затвора камеры.

В листинге 11.18 показан каркас, в котором создается и сохраняется снимок в формате JPEG на карту SD.

Листинг 11.18. Создание фотографии

```
private void takePicture() {
  camera.takePicture(shutterCallback, rawCallback, jpegCallback);
}

ShutterCallback shutterCallback = new ShutterCallback() {
  public void onShutter() {
    // TODO Действия при закрытии затвора камеры.
  }
};

PictureCallback rawCallback = new PictureCallback() {
  public void onPictureTaken(byte[] data, Camera camera) {
    // TODO Действия с необработанным изображением.
  }
};

PictureCallback jpegCallback = new PictureCallback() {
```

Продолжение ⇗

Листинг 11.18 (продолжение)

```
  public void onPictureTaken(byte[] data, Camera camera) {
    // Сохраните изображение в формате JPEG на карту SD
    FileOutputStream outStream = null;
    try {
      outStream = new FileOutputStream("/sdcard/test.jpg");
      outStream.write(data);
      outStream.close();
    } catch (FileNotFoundException e) {
      Log.d("CAMERA", e.getMessage());
    } catch (IOException e) {
      Log.d("CAMERA", e.getMessage());
    }
  }
};
```

Чтение и запись метаданных EXIF, хранящихся в изображении

Класс ExifInterface предоставляет механизм для чтения и изменения данных EXIF (Exchangeable Image File Format) внутри файла в формате JPEG. Создайте новый экземпляр этого класса, передав в его конструктор полное имя файла.

```
ExifInterface exif = new ExifInterface(filename);
```

Формат EXIF предназначен для хранения широкого спектра метаданных, описывающих фотографии, включая дату и время, свойства камеры (марку и модель), настройки изображения (апертуру и выдержку), а также описание снимка и информацию о местоположении.

Чтобы прочитать определенный атрибут из метаданных EXIF, вызовите метод getAttribute из объекта ExifInterface, передав ему имя нужного атрибута. Класс Exifinterface содержит статические константы вида TAG_*, которые можно использовать для доступа к стандартным данным EXIF. Чтобы изменить метаданные, примените метод setAttribute, передав ему в качестве параметров имя атрибута и соответствующее значение.

В листинге 11.19 показано, как из файла, хранящегося на карте SD, можно узнать координаты местности, где был сделан снимок, и модель камеры (информация о ней впоследствии отредактируется).

Листинг 11.19. Чтение и изменение метаданных EXIF

```
File file = new File(Environment.getExternalStorageDirectory(),
                     "test.jpg");
try {
  ExifInterface exif = new ExifInterface(file.getCanonicalPath());
  // Узнаем модель камеры и данные о местности
  String model = exif.getAttribute(ExifInterface.TAG_MODEL);
  float[] latLng = new float[2];
  exif.getLatLong(latLng);
```

```
    // Задаем марку камеры
    exif.setAttribute(ExifInterface.TAG_MAKE, "My Phone");
  } catch (IOException e) {
    Log.d("EXIF", e.getMessage());
  }
```

Добавление новых мультимедийных данных в MediaStore

По умолчанию мультимедийные файлы, созданные вашим приложением, недоступны для других программ. Выход из этой ситуации — добавление таких файлов в источник MediaStore, с помощью которого любое приложение может получить к ним доступ.

Android предоставляет два способа добавления данных в MediaStore: использование Сервиса MediaScannerConnection, который автоматически распознает и внесет ваш файл, или вручную добавление новой записи в соответствующий Источник данных.

Использование MediaScannerConnection

Если вы записали медиафайл, класс MediaScannerConnection поможет вам легко добавить его в MediaStore, избавляя от необходимости вручную создавать запись в этом Источнике данных.

Прежде чем использовать метод scanFile, чтобы начать сканирование файла, необходимо вызвать метод connect и дождаться установления связи с Сервисом MediaScannerConnection.

Этот вызов проходит в асинхронном режиме, поэтому вы должны реализовать интерфейс MediaScannerConnectionClient — оповестить приложение, что связь была установлена. Можно использовать этот же класс для уведомлений о завершении сканирования, чтобы знать, когда необходимо разорвать связь с MediaScannerConnection.

На самом деле все не так сложно, как выглядит. В листинге 11.20 показан каркас для создания нового класса MediaScannerConnectionClient, в котором определяется объект MediaScannerConnection, необходимый для добавления нового файла в источник MediaStore.

Листинг 11.20. Добавление файлов в MediaStore с помощью MediaScannerConnection

```
MediaScannerConnectionClient mediaScannerClient = new
MediaScannerConnectionClient() {
  private MediaScannerConnection msc = null;

  {
   msc = new MediaScannerConnection(getApplicationContext(), this);
```

Продолжение ⇗

Листинг 11.20 (продолжение)

```
     msc.connect();
    }

    public void onMediaScannerConnected() {
      msc.scanFile("/sdcard/test1.jpg", null);
    }

    public void onScanCompleted(String path, Uri uri) {
      msc.disconnect();
    }
  };
```

Добавление медиафайлов в MediaStore

Вместо того чтобы полагаться на MediaScannerConnection, вы сами можете добавить медиафайл в MediaStore, создав новый объект ContentValues и вставив его в соответствующий Источник данных.

В качестве метаданных для нового медиафайла нужно указывать название, временную отметку (timestamp) и географическую информацию, как показано в следующем фрагменте кода:

```
ContentValues content = new ContentValues(3);
content.put(Audio.AudioColumns.TITLE, "TheSoundandtheFury");
content.put(Audio.AudioColumns.DATE_ADDED,
            System.currentTimeMillis() / 1000);
content.put(Audio.Media.MIME_TYPE, "audio/amr");
```

Вы также должны указать абсолютный путь к файлу, который хотите добавить.

```
content.put(MediaStore.Audio.Media.DATA, "/sdcard/myoutputfile.mp4");
```

Получите доступ к объекту ContentResolver вашего приложения и используйте его для вставки новой строки в MediaStore, как показано ниже:

```
ContentResolver resolver = getContentResolver();
Uri uri = resolver.insert(MediaStore.Video.Media.EXTERNAL_CONTENT_URI,
                          content);
```

Как только медиафайл вставлен в MediaStore, вы должны объявить о его доступности с помощью Широковещательного намерения:

```
sendBroadcast(new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, uri));
```

Работа с несжатым звуком

Классы AudioTrack и AudioRecord позволяют напрямую записывать аудио и воспроизводить аудиопоток в формате PCM, используя аппаратные возможности устройства.

С помощью AudioTrack можно обрабатывать входящий аудиопоток и воспроизводить его практически в режиме реального времени, что дает возможность манипулировать входящим и исходящим звуком, обрабатывая на устройстве несжатые аудиоданные.

Хотя подробное описание обработки несжатого аудио выходит за рамки этой книги, следующие разделы познакомят с ходом записи и воспроизведения «сырых» данных в формате PCM.

Запись звука с помощью AudioRecord

Используйте класс AudioRecord для записи аудиоданных непосредственно с аппаратных буферов. Создайте новый объект AudioRecord, указав источник, частоту, настройки каналов, кодировщик для аудио и размер буфера.

```
int bufferSize = AudioRecord.getMinBufferSize(frequency,
                                              channelConfiguration,
                                              audioEncoding);

AudioRecord audioRecord = new AudioRecord(MediaRecorder.AudioSource.MIC,
                                          frequency,
channelConfiguration,
                                          audioEncoding, bufferSize);
```

Из соображений конфиденциальности Android требует, чтобы в манифесте вашего приложения значилось полномочие RECORD_AUDIO.

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

Значения частоты, кодировщика аудио и настроек каналов повлияют на размер и качество записанного аудиофайла. Обратите внимание, что ни один из этих параметров не связан с самими файлами.

Когда объект AudioRecord инициализирован, вызовите метод startRecording, чтобы начать асинхронную запись. Используйте метод read для добавления необработанных аудиоданных в записываемый буфер:

```
audioRecord.startRecording();
while (isRecording) {
  [ . . . заполнение буфера данными. . . ]
  int bufferReadResult = audioRecord.read(buffer, 0, bufferSize);
}
```

В листинге 11.21 демонстрируется запись несжатого аудио с микрофона с последующим сохранением в файл, размещенный на карте SD. Далее вы узнаете, как использовать AudioTrack, чтобы воспроизвести записанное.

Листинг 11.21. Запись необработанного звука с помощью AudioRecord

```
int frequency = 11025;
int channelConfiguration = AudioFormat.CHANNEL_CONFIGURATION_MONO;
int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
```

Продолжение ⇗

Листинг 11.21 (продолжение)

```
File file = new File(Environment.getExternalStorageDirectory(), "raw.pcm");

// Создайте новый файл.
try {
  file.createNewFile();
} catch (IOException e) {}

try {
  OutputStream os = new FileOutputStream(file);
  BufferedOutputStream bos = new BufferedOutputStream(os);
  DataOutputStream dos = new DataOutputStream(bos);

  int bufferSize = AudioRecord.getMinBufferSize(frequency,
                                               channelConfiguration,
                                               audioEncoding);
  short[] buffer = new short[bufferSize];

  // Создайте новый объект AudioRecord, чтобы записать звук.
  AudioRecord audioRecord = new AudioRecord(MediaRecorder.AudioSource.MIC,
                                           frequency,
                                           channelConfiguration,
                                           audioEncoding, bufferSize);
  audioRecord.startRecording();

  while (isRecording) {
    int bufferReadResult = audioRecord.read(buffer, 0, bufferSize);
    for (int i = 0; i < bufferReadResult; i++)
      dos.writeShort(buffer[i]);
  }

  audioRecord.stop();
  dos.close();
} catch (Throwable t) {}
```

Воспроизведение звука с помощью AudioTrack

Используйте класс AudioTrack, чтобы воспроизводить звук напрямую через аппаратные буферы устройства. Создайте новый объект AudioTrack, указав потоковый режим, частоту, параметры каналов, тип кодировщика и длину аудио.

```
AudioTrack audioTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
                                       frequency,
                                       channelConfiguration,
                                       audioEncoding,
                                       audioLength,
                                       AudioTrack.MODE_STREAM);
```

Поскольку аудиоданные необработанные, отсутствует метаинформация, связанная с ними. Поэтому важно установить корректные свойства, чтобы они совпадали с теми, которые были использованы при записи файла.

Инициализировав объект AudioTrack, вызовите метод play, чтобы начать асинхронное воспроизведение. Используйте метод write, чтобы добавить «сырые» аудиоданные в буфер проигрывателя.

```
audioTrack.play();
audioTrack.write(audio, 0, audioLength);
```

Вы можете начать запись в буфер объекта AudioTrack как до вызова метода play, так и после. В первом случае воспроизведение пойдет сразу после вызова, во втором — звук станет проигрываться, как только запишете данные в буфер AudioTrack.

В листинге 11.22 показано воспроизведение файла, записанного ранее, при этом ожидаемая частота аудиофайла удваивается, что приводит к увеличению скорости проигрывания вдвое.

Листинг 11.22. Воспроизведение звука с помощью AudioTrack

```
int frequency = 11025/2;
int channelConfiguration = AudioFormat.CHANNEL_CONFIGURATION_MONO;
int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
File file = new File(Environment.getExternalStorageDirectory(), "raw.pcm");

// Массив типа short для хранения аудиоданных (звук 16-битный,
// поэтому выделяем по 2 байта на значение)
int audioLength = (int)(file.length()/2);
short[] audio = new short[audioLength];

try {
  InputStream is = new FileInputStream(file);
  BufferedInputStream bis = new BufferedInputStream(is);
  DataInputStream dis = new DataInputStream(bis);

  int i = 0;
  while (dis.available() > 0) {
    audio[audioLength] = dis.readShort();
    i++;
  }

  // Закрытие входящих потоков.
  dis.close();

  // Создание объекта AudioTrack и проигрывание звука с его помощью
  AudioTrack audioTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
                                         frequency,
                                         channelConfiguration,
                                         audioEncoding,
                                         audioLength,
                                         AudioTrack.MODE_STREAM);
  audioTrack.play();
  audioTrack.write(audio, 0, audioLength);
} catch (Throwable t) {}
```

Распознавание речи

Начиная с версии 1.5 (API level 3) Android поддерживает голосовой ввод и распознавание речи с помощью класса RecognizerIntent.

Этот API позволяет встраивать в приложения голосовой ввод, используя стандартное диалоговое окно, как показано на рис. 11.1.



Рис. 11.1.

Речь распознается с помощью метода startNewActivityForResult, которому необходимо передать Намерение, созданное с применением константы RecognizerIntent.ACTION_RECOGNIZE_SPEECH.

Намерение также должно содержать дополнительный параметр RecognizerIntent.EXTRA_LANGUAGE_MODEL, указывающий на языковую модель распознавания входящего звука. Он может содержать следующие значения: LANGUAGE_MODEL_FREE_FORM или LANGUAGE_MODEL_WEB_SEARCH. Оба доступны в виде статических констант из класса RecognizerIntent.

Еще вы можете указать несколько необязательных параметров, чтобы изменять язык, количество потенциальных результатов и содержа-

ние отображаемой строки, используя следующие константы из класса RecognizerIntent:

- EXTRA_PROMPT — задайте строку, которая будет отображаться в окне голосового ввода (показано на рис. 11.1). Она должна предлагать пользователю сказать что-нибудь в микрофон;
- EXTRA_MAXRESULTS — используйте целочисленное значение, чтобы ограничить количество результатов, которые могут быть возвращены при распознавании;
- EXTRA_LANGUAGE — укажите языковую константу из класса Locale, чтобы задать язык для ввода, отличный от языка по умолчанию на данном устройстве. Вы можете получить текущее значение по умолчанию, вызвав статический метод getDefault из класса Locale.

ВНИМАНИЕ

Движок, отвечающий за распознавание речи, может не справиться со всеми языками, доступными в классе Locale.

Не все устройства имеют поддержку распознавания речи. В таких случаях, как правило, можно загрузить соответствующую библиотеку из Android Market.

В листинге 11.23 показывается, как инициировать распознавание речи на английском языке, используя собственную строку в диалоговом окне и ограничивая количество результатов до одного.

Листинг 11.23. Создание диалогового окна для распознавания речи

```
Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
// Укажите свободную форму ввода
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
                "or forever hold your peace");
intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 1);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.ENGLISH);
startActivityForResult(intent, VOICE_RECOGNITION);
```

Когда пользователь завершит голосовой ввод, записанный звук проанализируется и обработается движком для распознавания речи. Результаты вернутся через обработчик onActivityResult в виде списка строк, хранящегося в параметре EXTRA_RESULTS, как показано в листинге 11.24.

Листинг 11.24. Извлечение результатов распознавания речи

```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode,
                                Intent data) {
```

Продолжение ☞

Листинг 11.24 (продолжение)

```
  if (requestCode == VOICE VOICE_RECOGNITION && resultCode == RESULT_OK)
{
    ArrayList<String> results;
    results = data.getStringArrayListExtra(RecognizerIntent.EXTRA_
RESULTS);
    // TODO Выполнить какие-то действия с распознанными строками
  }
  super.onActivityResult(requestCode, resultCode, data);
}
```

Резюме

В этой главе вы узнали, как проигрывать, записывать и захватывать мультимедийные данные внутри своего приложения.

Начав с объекта MediaPlayer, научились воспроизводить аудио- и видеоданные из локальных файлов, ресурсов приложения и удаленных веб-сайтов, поддерживающих потоковое вещание. Познакомились с элементом VideoView и узнали, как создавать объекты SurfaceView, чтобы проигрывать видеоданные, предоставлять функцию предварительного просмотра и отображать живой видеопоток из камеры.

Вы научились использовать Намерения для запуска стандартных приложений, чтобы записывать видео и делать снимки. Используя классы MediaRecorder и Camera, реализовали собственные компоненты для захвата статических изображений и подвижных сцен.

Продемонстрировано также, как читать и изменять данные EXIF, принадлежащие изображению, как добавлять новые медиафайлы в источник MediaStore и работать с несжатыми аудиоданными.

В завершение познакомились с библиотеками распознавания речи и голоса, научились использовать их для добавления поддержки голосового ввода в собственные приложения.

В следующей главе вы изучите низкоуровневые API для связи, доступные на платформе Android, научитесь применять API в Android, отвечающие за телефонию, чтобы отслеживать состояние подключения к мобильной сети, звонки и SMS. Узнаете, как использовать API для исходящих звонков и чтобы отсылать и принимать текстовые сообщения внутри своего приложения.

Глава 12
ТЕЛЕФОНИЯ И SMS

Содержание главы

- Инициирование телефонных звонков.
- Считывание состояния телефонного соединения, подключения к Интернету и передачи данных, а также статуса SIM-карты.
- Отслеживание состояния телефонного соединения, подключения к Интернету и передачи данных, а также статуса SIM-карты.
- Использование `Намерений` для передачи сообщений — SMS (Short Messaging Service) и MMS (Multimedia Messaging Service).
- Использование объекта SmsManager для передачи SMS.
- Обработка входящих SMS.

В этой главе вы узнаете, как использовать API Android для телефонии, чтобы отслеживать мобильные соединения, передачу данных, входящие и исходящие звонки, научитесь передавать и принимать SMS.

Вы получите представление о коммуникационном аппаратном обеспечении, исследуя пакет, обеспечивающий поддержку телефонии. Данный пакет предназначен для мониторинга состояния телефона и телефонных звонков, вызовов и чтения информации о входящих звонках.

Android также предоставляет полный доступ к возможностям, связанным с SMS, позволяет отсылать и принимать текстовые сообщения внутри собственного приложения. Используя эти API, вы можете создать собственное клиентское приложение для работы с SMS, заменив стандартные программы, входящие в дистрибутив. У вас будет возможность добавлять поддержку текстовых сообщений в собственные проекты, привнося в них социальные возможности, работающие поверх протокола SMS.

В конце данной главы запланировано использовать объект SmsManager в приложении, которое способно отвечать на экстренные сообщения посредством SMS. В критических ситуациях эта программа позволит пользователям быстро (или даже автоматически) отвечать людям, которые поинтересуются их безопасностью.

Телефония

API в Android, предназначенные для поддержки телефонии, открывают доступ приложений к низкоуровневому аппаратному стеку телефона. Благодаря этому разработчики могут инициировать набор номера или интегрировать обработку звонков и отслеживание состояния телефона в свои проекты.

ВНИМАНИЕ

Из соображений безопасности текущая версия Android SDK не позволяет создавать собственные Активности, которые отображаются при входящих или исходящих звонках.

В следующих разделах рассматривается возможность отслеживания и управления телефоном, услугами связи и событиями, связанными с сотовой сетью. Это позволит вашим приложениям управлять стандартными телефонными функциями и даже расширит их возможности.

Телефонные звонки

Для инициирования телефонных звонков используются Намерения, запускающие для этого стандартное приложение. Вызывая Активность для дозвона с помощью Намерения, необходимо указать номер, на который пойдет звонок, используя схему tel: в качестве компонента с данными.

Вместо немедленного дозвона примените действие Intent.ACTION_DIAL, чтобы отобразить экран для набора номера. Оно загружает Активность для дозвона и передает ей указанный номер, позволяя стандартному приложению самому управлять инициализацией вызова (системное телефонное приложение предлагает решать пользователю, делать вызов или нет). Этот подход не требует никаких полномочий и считается стандартным путем осуществления звонков из сторонних приложений.

В листинге 12.1 демонстрируется базовая методика для набора телефонного номера.

Листинг 12.1. Набор номера

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:1234567"));
startActivity(intent);
```

Используя Намерение для того, чтобы объявить необходимость набрать номер, ваше приложение может оставаться независимым от реализации программы, которая занимается непосредственно дозвоном. Например, если вы на своем устройстве заменили стандартное приложение на гибридное, совместимое с IP-телефонией, вышеописанный подход позволит другим приложениям получить доступ к новым возможностям.

Замена стандартного приложения для дозвона

Замена стандартного приложения для дозвона проходит в два этапа:

1) перехват Намерений, которые в настоящее время обслуживаются стандартным приложением;

2) осуществление исходящих звонков и при необходимости управление ими.

Пытаясь отобразить номер телефона или посылая запрос на звонок с помощью схемы tel:, стандартная телефонная программа реагирует на действия Намерений, которые вызываются при нажатии пользователем аппаратной кнопки дозвона.

Чтобы перехватывать подобные запросы, добавьте в описание своей новой Активности тег `<intent-filter>`, который отслеживает следующие действия.

- Intent.ACTION_CALL_BUTTON. Передается в момент, когда нажимается аппаратная кнопка для звонка. Создайте Фильтр намерений, изначально отслеживающий данное действие.
- Intent.ACTION_DIAL. Это действие описывалось в предыдущем разделе. Оно используется приложениями, которые хотят запустить программу для дозвона. Фильтр намерений, работающий с данным дейсвием, должен иметь сразу две категории: DEFAULT и BROWSABLE (чтобы поддерживать запросы на дозвон из браузера), а также содержать схему tel:, чтобы разрешить замену существующих телефонных функций (могут использоваться и дополнительные схемы).
- Intent.ACTION_VIEW. Используется приложениями для отображения какого-либо набора данных. Не забудьте указать в Фильтре намерений схему tel:, чтобы задействовать новую Активность для отображения телефонных номеров.

В следующем манифесте описывается Активность с Фильтрами намерений, которые перехватывают вышеописанные действия:

```
<activity
  android:name=".MyDialerActivity"
  android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.CALL_BUTTON" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.DIAL" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="tel" />
  </intent-filter>
</activity>
```

Когда приложение запустится, у пользователей появится возможность вводить или изменять телефонный номер, а также совершать исходящие звонки.

Наиболее простой подход — использование существующих телефонных возможностей. В этом случае вы можете применить действие Intent.ACTION_CALL, запустив стандартную Активность для дозвона и позволив системе управлять набором номера, устанавливать связь и передавать голосовые данные. Ваше приложение должно содержать полномочие CALL_PHONE, чтобы инициировать это действие.

В качестве альтернативного варианта можно полностью заменить исходный телефонный стек, реализовав собственный фреймворк для поддержки дозвона и передачи голосовых данных. Такой подход идеален при создании приложения с поддержкой IP-телефонии. Имейте в виду, что описание реализации альтернативной телефонной платформы выходит за рамки данной книги.

Вы можете перехватывать вышеописанные Намерения для внесения изменений в исходящие телефонные звонки или для их блокировки, необязательно реализовывать полноценный аналог приложения для дозвона.

Доступ к свойствам телефона и сети, а также отслеживание подключения и операций по передаче данных

Доступ к телефонным API контролируется объектом TelephonyManager, который можно получить с помощью метода getSystemService, как показано в листинге 12.2.

Листинг 12.2. Доступ к объекту TelephonyManager

```
String srvcName = Context.TELEPHONY_SERVICE;
TelephonyManager telephonyManager = (TelephonyManager)
getSystemService(srvcName);
```

TelephonyManager предоставляет прямой доступ ко многим телефонным свойствам, в том числе к информации об устройстве, сети, SIM-карте и ходе передачи данных.

Считывание информации о телефонном устройстве

С помощью объекта TelephonyManager можно получить тип телефона (GSM или CDMA), уникальный идентификатор (IMEI или MEID), версию программного обеспечения, а также телефонный номер. Обратите внимание, что все эти свойства (кроме первого) требуют наличия в манифесте приложения полномочия READ_PHONE_STATE.

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

В листинге 12.3 показано, как извлечь эту информацию.

Листинг 12.3. Считывание информации о телефонном устройстве

```
int phoneType = telephonyManager.getPhoneType();
switch (phoneType) {
  case (TelephonyManager.PHONE_TYPE_CDMA): break;
  case (TelephonyManager.PHONE_TYPE_GSM) : break;
  case (TelephonyManager.PHONE_TYPE_NONE): break;
  default: break;
}

// -- Эти свойства требуют наличие полномочия READ_PHONE_STATE --
// Считывание IMEI для GSM или MEID для CDMA
String deviceId = telephonyManager.getDeviceId();
// Считывание версии программного обеспечения на телефоне (учтите,
// что это не версия SDK)
String softwareVersion = telephonyManager.getDeviceSoftwareVersion();
// Получение телефонного номера
String phoneNumber = telephonyManager.getLine1Number();
```

Считывание информации о состоянии соединения и о статусе процесса передачи данных

С помощью методов getDataState и getDataActivity можно узнать текущее состояние соединения с сетью и информацию о передаче данных, как показано в листинге 12.4. Далее в этой главе вы научитесь отслеживать изменения этих параметров.

Листинг 12.4. Считывание информации о состоянии соединения и статусе процесса передачи данных

```
int dataActivity = telephonyManager.getDataActivity();
int dataState = telephonyManager.getDataState();

switch (dataActivity) {
  case TelephonyManager.DATA_ACTIVITY_IN : break;
  case TelephonyManager.DATA_ACTIVITY_OUT : break;
  case TelephonyManager.DATA_ACTIVITY_INOUT : break;
  case TelephonyManager.DATA_ACTIVITY_NONE : break;
}

switch (dataState) {
  case TelephonyManager.DATA_CONNECTED : break;
  case TelephonyManager.DATA_CONNECTING : break;
  case TelephonyManager.DATA_DISCONNECTED : break;
  case TelephonyManager.DATA_SUSPENDED : break;
}
```

Считывание информации о телефонной сети

Подключившись к сети, используйте объект TelephonyManager для считывания кодов страны и сети (MCC+MNC), национального кода в формате ISO, а также типа данной сети. Эту информацию можно получить только

при установленном соединении с мобильной сетью. Она может оказаться недостоверной, если подключение осуществляется по CDMA. Используйте метод getPhoneType, как показано ниже, чтобы узнать, к сети какого типа вы подключены.

В листинге 12.5 показано, как извлечь информацию о сети, а также список поддерживаемых стандартов.

Листинг 12.5. Считывание информации о телефонной сети

```
// Получите код страны, к чьей сети вы подключены
String networkCountry = telephonyManager.getNetworkCountryIso();
// Получите идентификатор оператора сети (MCC + MNC)
String networkOperatorId = telephonyManager.getNetworkOperator();
// Получите название оператора сети
String networkName = telephonyManager.getNetworkOperatorName();
// Получите тип сети, к которой вы подключены
int networkType = telephonyManager.getNetworkType();
switch (networkType) {
  case (TelephonyManager.NETWORK_TYPE_1xRTT)   : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_CDMA)    : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_EDGE)    : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_EVDO_0)  : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_EVDO_A)  : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_GPRS)    : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_HSDPA)   : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_HSPA)    : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_HSUPA)   : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_UMTS)    : [ ... какие-то
                                                 действия ... ]
                                                 break;
  case (TelephonyManager.NETWORK_TYPE_UNKNOWN) : [ ... какие-то
                                                 действия ... ]
                                                 break;
  default: break;
}
```

Считывание данных с SIM-карты

Если устройство, на котором работает ваше приложение, обладает поддержкой GSM, оно должно иметь SIM-карту. Данные, которые вы можете извлечь из нее с помощью объекта TelephonyManager, — код страны в формате ISO, название оператора, коды MCC (мобильный код страны) и MNC (код мобильной сети). Эта информация пригодится, если потребуется предоставить особые возможности конкретному оператору.

Вы также можете узнать серийный номер текущей SIM-карты, если добавите в манифест своего приложения полномочие READ_PHONE_STATE.

Прежде чем начать работать с данными методами, необходимо убедиться, что SIM-карта в состоянии готовности. Узнать это можно с помощью метода getSimState, как показано в листинге 12.6.

Листинг 12.6. Считывание данных с SIM-карты

```
int simState = telephonyManager.getSimState();
switch (simState) {
  case (TelephonyManager.SIM_STATE_ABSENT): break;
  case (TelephonyManager.SIM_STATE_NETWORK_LOCKED): break;
  case (TelephonyManager.SIM_STATE_PIN_REQUIRED): break;
  case (TelephonyManager.SIM_STATE_PUK_REQUIRED): break;
  case (TelephonyManager.SIM_STATE_UNKNOWN): break;
  case (TelephonyManager.SIM_STATE_READY): {
    // Извлеките из SIM-карты код страны в формате ISO
    String simCountry = telephonyManager.getSimCountryIso();
    // Получите код оператора активной SIM-карты (MCC + MNC)
    String simOperatorCode = telephonyManager.getSimOperator();
    // Извлеките из SIM-карты название оператора
    String simOperatorName = telephonyManager.getSimOperatorName();
    // -- Требуется наличие полномочия READ_PHONE_STATE --
    // Получите серийный номер SIM-карты
    String simSerial = telephonyManager.getSimSerialNumber();
    break;
  }
  default: break;
}
```

Отслеживание изменений в состоянии подключения к сети, статусе телефона и телефонной активности

API в Android, отвечающий за телефонию, позволяет следить за состоянием телефона, получать телефонные номера, из которых идут входящие звонки, наблюдать за изменениями уровня сигнала, статуса сетевого соединения и передачей данных.

Чтобы отслеживать и контролировать состояние телефона, манифест вашего приложения должен содержать полномочие READ_PHONE_STATE, как показано в следующем фрагменте кода:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

Путем наследования класса PhoneStateListener вы можете отслеживать изменения состояния телефона (и реагировать на них), включая такие показатели, как статус вызова («звонок», «трубка снята» и т. д.), местоположение относительно базовых станций, статус голосовой почты и переадресации вызовов, телефонные услуги и уровень сотового сигнала.

Чтобы реагировать на события, возникающие при изменении состояния телефона, создайте новую реализацию класса PhoneStateListener и переопределите обработчики тех событий, которые вас интересуют. Каждый обработчик принимает параметры, описывающие новое состояние телефона (например, местоположение относительно базовых станций, состояние сотового соединения или уровень сигнала).

В листинге 12.7 показан каркас реализации класса PhoneStateListener, в котором перечислены доступные обработчики для отслеживания изменений состояния телефона.

Листинг 12.7. Каркас реализации PhoneStateListener

```
PhoneStateListener phoneStateListener = new PhoneStateListener() {
  public void onCallForwardingIndicatorChanged(boolean cfi) {}
  public void onCallStateChanged(int state, String incomingNumber) {}
  public void onCellLocationChanged(CellLocation location) {}
  public void onDataActivity(int direction) {}
  public void onDataConnectionStateChanged(int state) {}
  public void onMessageWaitingIndicatorChanged(boolean mwi) {}
  public void onServiceStateChanged(ServiceState serviceState) {}
  public void onSignalStrengthChanged(int asu) {}
};
```

Создав собственный объект класса PhoneStateListener, зарегистрируйте его с помощью TelephonyManager, используя битовую маску для определения событий, которые вы хотите отслеживать, как показано в листинге 12.8.

Листинг 12.8. Регистрация PhoneStateListener

```
telephonyManager.listen(phoneStateListener,
                        PhoneStateListener.LISTEN_CALL_FORWARDING_
INDICATOR |
                        PhoneStateListener.LISTEN_CALL_STATE |
                        PhoneStateListener.LISTEN_CELL_LOCATION |
                        PhoneStateListener.LISTEN_DATA_ACTIVITY |
                        PhoneStateListener.LISTEN_DATA_CONNECTION_STATE |
                        PhoneStateListener.LISTEN_MESSAGE_WAITING_
INDICATOR |
                        PhoneStateListener.LISTEN_SERVICE_STATE |
                        PhoneStateListener.LISTEN_SIGNAL_STRENGTH);
```

Чтобы отменить регистрацию, вызовите метод listen, передав ему битовое поле PhoneStateListener.LISTEN_NONE в качестве параметра, как показано ниже:

```
telephonyManager.listen(phoneStateListener,
                        PhoneStateListener.LISTEN_NONE);
```

Отслеживание входящих телефонных звонков

Одна из частых причин отслеживания состояния телефона — необходимость обнаружения входящих телефонных звонков и реакции на них.

Для этого следует переопределить метод onCallStateChanged из реализации класса PhoneStateListener и зарегистрировать соответствующее событие, как показано в листинге 12.9. После этого вы будете получать уведомления, если состояние вызова изменится.

Листинг 12.9. Отслеживание телефонных звонков

```
PhoneStateListener callStateListener = new PhoneStateListener() {
  public void onCallStateChanged(int state, String incomingNumber) {
    // TODO реакция на входящий звонок.
  }
};

telephonyManager.listen(callStateListener,
                        PhoneStateListener.LISTEN_CALL_STATE);
```

Обработчик onCallStateChanged принимает телефонный номер входящего звонка, а также параметр, описывающий текущее состояние звонка. Он может иметь одно из трех значений:

- TelephonyManager.CALL_STATE_IDLE — когда телефон не находится в состояниях дозвона или разговора;
- TelephonyManager.CALL_STATE_RINGING — когда телефон дозванивается;
- TelephonyManager.CALL_STATE_OFFHOOK — когда телефон находится в состоянии разговора.

Отслеживание изменений местоположения устройства относительно базовых станций

Вы можете получать уведомления каждый раз, когда изменяется текущее местоположение устройства относительно базовых станций. Для этого необходимо переопределить метод onCellLocationChanged из реализации класса PhoneStateListener. Прежде чем зарегистрировать данную функцию, добавьте в манифест своего приложения полномочие ACCESS_COARSE_LOCATION.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Обработчик onCellLocationChanged принимает в качестве параметра объект CellLocation, содержащий методы, с помощью которых можно извлечь Cell ID (getCid) и код текущей локальной зоны LAC (getLac).

В листинге 12.10 показано, как с помощью объекта PhoneStateListener отслеживать изменения местоположения устройства относительно базовых станций, отображая уведомление типа Toast, содержащее новый идентификатор Cell ID.

Листинг 12.10. Отслеживание смены базовых станций

```
PhoneStateListener cellLocationListener = new PhoneStateListener() {
  public void onCellLocationChanged(CellLocation location) {
    GsmCellLocation gsmLocation = (GsmCellLocation)location;
    Toast.makeText(getApplicationContext(),
                   String.valueOf(gsmLocation.getCid()),
                   Toast.LENGTH_LONG).show();
  }
};
telephonyManager.listen(cellLocationListener,
                        PhoneStateListener.LISTEN_CELL_LOCATION);
```

Отслеживание состояния услуг телефонии

Обработчик onServiceStateChanged следит за информацией о телефонных услугах. Используйте параметр ServiceState, чтобы узнать текущее состояние услуги.

Метод getState из объекта ServiceState возвращает текущее состояние услуги в виде одной из констант:

- STATE_IN_SERVICE — телефонные услуги доступны в обычном режиме;
- STATE_EMERGENCY_ONLY — телефонные услуги доступны, но только для вызова служб экстренной помощи;
- STATE_OUT_OF_SERVICE — в настоящее время телефонная сеть недоступна;
- STATE_POWER_OFF — телефонный модуль выключен (обычно, когда активизирован «режим полета»).

С помощью методов вида getOperator* вы можете получать информацию об операторе, предоставляющем услуги телефонии. Используя метод getRoaming, можно узнать, находится ли устройство в режиме роуминга.

В листинге 12.11 показано, как зарегистрировать обработчик, чтобы отслеживать изменения состояния телефонных услуг и отображать уведомление типа Toast, содержащее название оператора, который их предоставляет.

Листинг 12.11. Отслеживание состояния услуг телефонии

```
PhoneStateListener serviceStateListener = new PhoneStateListener() {
  public void onServiceStateChanged(ServiceState serviceState) {
    if (serviceState.getState() == ServiceState.STATE_IN_SERVICE) {
      String toastText = serviceState.getOperatorAlphaLong();
      Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_
SHORT);
    }
  }
};

telephonyManager.listen(serviceStateListener,
                        PhoneStateListener.LISTEN_SERVICE_STATE);
```

Отслеживание подключения к сети для передачи данных

Как и в случае с информацией о звонках и телефонных услугах, с помощью объекта PhoneStateListener вы можете следить за изменениями в состоянии подключения к сети для передачи данных (и за самой передачей).

Для этих целей PhoneStateListener содержит два обработчика событий. Переопределите метод onDataActivity, чтобы следить за передачей данных, и onDataConnectionStateChanged, чтобы узнавать об изменениях в состоянии подключения к сети, по которой эти данные передаются.

В листинге 12.12 переопределяются оба эти обработчика. С помощью оператора switch демонстрируются все возможные значения для параметров state и direction, которые передаются при каждом из этих событий.

Листинг 12.12. Отслеживание состояния передачи данных и подключения к сети

```
PhoneStateListener dataStateListener = new PhoneStateListener() {
  public void onDataActivity(int direction) {
    switch (direction) {
      case TelephonyManager.DATA_ACTIVITY_IN : break;
      case TelephonyManager.DATA_ACTIVITY_OUT : break;
      case TelephonyManager.DATA_ACTIVITY_INOUT : break;
      case TelephonyManager.DATA_ACTIVITY_NONE : break;
    }
  }

  public void onDataConnectionStateChanged(int state) {
    switch (state) {
      case TelephonyManager.DATA_CONNECTED : break;
      case TelephonyManager.DATA_CONNECTING : break;
      case TelephonyManager.DATA_DISCONNECTED : break;
      case TelephonyManager.DATA_SUSPENDED : break;
    }
  }
};

telephonyManager.listen(dataStateListener,
                        PhoneStateListener.LISTEN_DATA_ACTIVITY |
                        PhoneStateListener.LISTEN_DATA_CONNECTION_STATE);
```

Знакомство с SMS и MMS

Если вашему телефону еще нет двадцати лет, скорее всего, вы уже знаете, что такое SMS. Сегодня это одна из наиболее востребованных функций в мобильных телефонах. Многие пользователи предпочитают набирать текстовые сообщения, вместо того чтобы звонить.

Технология SMS разработана для обмена небольшой по объему текстовой информацией между мобильными телефонами. Она поддерживает отправку как текстовых (пригодных для чтения человеком), так и бинарных (созданных для обработки приложениями) сообщений. Более современный протокол под названием MMS позволяет пользователям отправлять и получать сообщения с мультимедийными данными: фотографиями, видео- и аудиофайлами.

Поскольку обе эти технологии неновые, издано немало пособий, описывающих технические нюансы устройства SMS и MMS, а также процесс передачи их по сети. Вместо того чтобы все это пересказывать, в следующих разделах сосредоточимся на практических аспектах отправки и получения текстовых, бинарных и мультимедийных сообщений внутри приложений для Android.

Использование SMS и MMS в вашем приложении

Android предоставляет разработчикам приложений полную поддержку SMS. С помощью объекта SmsManager вы можете заменить стандартную программу для работы с SMS, отсылая и принимая текстовые сообщения или используя SMS в виде протокола для доставки данных.

В настоящее время API в Android не поддерживает создание сообщений MMS внутри сторонних приложений, но у вас есть возможность использовать действия SEND и SEND_TO внутри намерений, чтобы с помощью системной программы, установленной на устройстве, отправлять как SMS, так и MMS.

В этой главе вы узнаете, как использовать SmsManager и Намерения для передачи сообщений из своего приложения.

Доставляются SMS отнюдь не мгновенно. Если сравнивать с протоколами, основанными на IP и сокетах, передача данных между приложениями по SMS отличается низкой скоростью, потенциальной дороговизной и высоким уровнем латентности. Как результат, SMS совершенно не подходит для задач, которые требуют отзывчивости, близкой к режиму реального времени.

Тем не менее широкое распространение и гибкость сетей для передачи SMS делает этот вид обмена данными отличным инструментом доставки информации пользователям, на телефонах которых не установлена операционная система Android, что уменьшает зависимость от сторонних серверов.

Передача SMS и MMS из вашего приложения с помощью Намерений и стандартного клиента

Иногда проще доверить другому приложению всю работу по передаче SMS и MMS вместо того, чтобы самостоятельно реализовывать полноценный клиент для этих задач внутри своей программы.

Чтобы это сделать, вызовите метод startActivity с помощью Намерения, которому присвоено действие Intent.ACTION_SENDTO. В качестве данных для Намерения укажите номер адресата, используя схему sms:. С помощью дополнительного параметра sms_body задайте текст сообщения. Все эти шаги продемонстрированы в листинге 12.13.

Листинг 12.13. Передача SMS с помощью Намерения

```
Intent smsIntent = new Intent(Intent.ACTION_SENDTO,
                              Uri.parse("sms:55512345"));
smsIntent.putExtra("sms_body", "Press send to send me");
startActivity(smsIntent);
```

Вы также можете прикреплять файлы (фактически, создавая MMS) к своим сообщениям. Добавьте параметр Intent.EXTRA_STREAM, ассоциировав его с путем URI к вложению, и присвойте параметру type тип MIME прикрепленного ресурса.

Обратите внимание, что стандартное приложение для работы с MMS не содержит Фильтра намерений для действия ACTION_SENDTO с установкой параметра type. Вместо этого необходимо использовать действие ACTION_SEND, а также добавить телефонный номер в качестве дополнительного параметра address, как показано в листинге 12.14.

Листинг 12.14. Отправка MMS с прикрепленным изображением

```
// Получите путь URI к данным, которые нужно прикрепить.
Uri attached_Uri = Uri.parse("content://media/external/images/media/1");

// Создайте новое Намерение для передачи MMS
Intent mmsIntent = new Intent(Intent.ACTION_SEND, attached_Uri);
mmsIntent.putExtra("sms_body", "Please see the attached image");
mmsIntent.putExtra("address", "07912355432");
mmsIntent.putExtra(Intent.EXTRA_STREAM, attached_Uri);
mmsIntent.setType("image/png");
startActivity(mmsIntent);
```

ПРИМЕЧАНИЕ

При запуске примера, показанного в листинге 12.14, пользователю, вероятно, будет предложено выбрать одно из нескольких приложений, способных выполнять запросы подобного рода, включая Gmail, клиент для работы с e-mail и программу для отправки SMS.

Отправка SMS вручную

За работу с SMS в Android отвечает класс SmsManager. Вы можете получить ссылку на объект этого класса, используя статический метод SmsManager.getDefault, как показано в следующем фрагменте кода.

```
SmsManager smsManager = SmsManager.getDefault();
```

ПРИМЕЧАНИЕ

В более ранних версиях Android, предшествовавших 1.6 (SDK level 4), классы SmsManager и SmsMessage предоставлялись пакетом android.telephony.gsm. Теперь такое размещение считается устаревшим и данные классы хранятся в пакете android.telephony, обеспечивая тем самым поддержку устройств GSM и CDMA.

Для передачи SMS приложения должны обладать полномочием SEND_SMS. Поэтому необходимо добавить в манифест следующую строку:

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Отправка текстовых сообщений

Для того чтобы отправить текстовое сообщение, используйте метод sendTextMessage из объекта SmsManager, передавая ему адрес (телефонный номер) получателя и текст послания, как показано в листинге 12.15.

Листинг 12.15. Передача SMS

```
String sendTo = "5551234";
String myMessage = "Android supports programmatic SMS messaging!";

smsManager.sendTextMessage(sendTo, null, myMessage, null, null);
```

Второй параметр может быть использован для указания центра обработки сообщений (SMSC). Если вы передали значение null, как это продемонстрировано в листинге 12.15, будет использован стандартный центр, предоставляемый вашим оператором.

Последние два параметра позволяют задать Намерения для отслеживания передачи и успешной доставки вашего сообщения.

Чтобы реагировать на срабатывание этих Намерений, создайте и зарегистрируйте Приемники широковещательных намерений, как показано в следующих разделах.

ПРИМЕЧАНИЕ

Android Debug Bridge (ADB) поддерживает передачу SMS между несколькими экземплярами эмулятора. Чтобы отправить SMS из одного

эмулятора в другой, укажите номер порта получателя в качестве первого параметра для метода sendTextMessage.

Android автоматически адресует ваше сообщение соответствующему экземпляру эмулятора, и оно будет обработано, как обычное SMS.

Отслеживание и подтверждение доставки SMS

Чтобы отслеживать и подтверждать успешность доставки своих исходящих SMS, реализуйте и зарегистрируйте Приемники широковещательных намерений. Они должны быть настроены на действия, указанные вами при создании Ожидающих намерений, которые вы передали в метод sendTextMessage.

Первое Намерение, sentIntent, срабатывает, когда сообщение успешно отправляется (либо когда возникают какие-то проблемы). Код результата для Приемника, который получает данное Намерение, будет иметь одно из следующих значений:

- Activity.RESULT_OK — успешная отправка;
- SmsManager.RESULT_ERROR_GENERIC_FAILURE — при отправке возникли неизвестные проблемы;
- SmsManager.RESULT_ERROR_RADIO_OFF — телефонный модуль выключен;
- SmsManager.RESULT_ERROR_NULL_PDU — возникла проблема, связанная с форматом PDU (protocol description unit).

Второе Намерение, deliveryIntent, срабатывает только после того, как адресат получил ваше сообщение.

В листинге 12.16 показан типичный шаблон для отправки SMS и отслеживания успешности его передачи и доставки.

Листинг 12.16. Отслеживание доставки SMS

```
String SENT_SMS_ACTION = "SENT_SMS_ACTION";
String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION";

// Создайте параметр sentIntent
Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI = PendingIntent.getBroadcast(getApplicationContext(),
                                                   0,
                                                   sentIntent,
                                                   0);

// Создайте параметр deliveryIntent
Intent deliveryIntent = new Intent(DELIVERED_SMS_ACTION);
PendingIntent deliverPI =
  PendingIntent.getBroadcast(getApplicationContext(),
```

Продолжение ⇨

Листинг 12.16 (продолжение)

```
                                0,
                                deliveryIntent,
                                0);

// Зарегистрируйте Приемники Широковещательных Намерений
registerReceiver(new BroadcastReceiver() {
                   @Override
                   public void onReceive(Context _context, Intent _
intent)
                   {
                     switch (getResultCode()) {
                       case Activity.RESULT_OK:
                          [ . . . действия при успешной отправке . . . ];
                          break;
                       case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                          [ . . . действия при неизвестных
                          проблемах . . . ]; break;
                       case SmsManager.RESULT_ERROR_RADIO_OFF:
                          [ . . . действия при выключенном телефонном
                          модуле . . . ]; break;
                       case SmsManager.RESULT_ERROR_NULL_PDU:
                          [ . . . действия при возникновении проблем,
                          связанных с форматом PDU . . . ]; break;
                     }
                   }
                 },
                 new IntentFilter(SENT_SMS_ACTION));

registerReceiver(new BroadcastReceiver() {
                   @Override
                   public void onReceive(Context _context, Intent _intent)
                   {
                     [ . . . Действия при успешной доставке SMS . . . ]
                   }
                 },
                 new IntentFilter(DELIVERED_SMS_ACTION));

// Отправьте сообщение
smsManager.sendTextMessage(sendTo, null, myMessage, sentPI, deliverPI);
```

Проверка SMS на соответствие максимальной допустимой длине

Длина текста в SMS, как правило, ограничена 160 символами, поэтому сообщения, которые выходят за эти рамки, должны быть разбиты на несколько более мелких частей. Класс SmsManager содержит метод divideMessage, который принимает в качестве параметра строку и делит ее на массив с сообщениями, каждое из которых имеет допустимый размер.

Затем вы можете использовать метод sendMultipartTextMessage из SmsManager, чтобы отослать полученный массив, как показано в листинге 12.17.

sentIntent и deliveryIntent в методе sendMultipartTextMessage — массив, который может содержать разные Намерения, срабатывающие для конкретных частей сообщения.

Листинг 12.17. Передача длинных сообщений несколькими кусками

```
ArrayList<String> messageArray = smsManager.divideMessage(myMessage);
ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>();
for (int i = 0; i < messageArray.size(); i++)
  sentIntents.add(sentPI);

smsManager.sendMultipartTextMessage(sendTo,
                                    null,
                                    messageArray,
                                    sentIntents, null);
```

Передача бинарных сообщений

Вы можете отправлять через SMS бинарные данные, используя метод sendDataMessage из объекта SmsManager. Работает он по тому же принципу, что и sendTextMessage, но включает дополнительные параметры для указания порта адресата и массива байтов, который содержит данные для отправки.

В листинге 12.18 показан базовый принцип отправки бинарного сообщения.

Листинг 12.18. Отправка бинарных SMS

```
Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI = PendingIntent.getBroadcast(getApplicationContext(),
                                                  0, sentIntent, 0);

short destinationPort = 80;
byte[] data = [ . . . ваши данные . . . ];
smsManager.sendDataMessage(sendTo, null, destinationPort,
                           data, sentPI, null);
```

Отслеживание входящих SMS

Когда устройство принимает новое SMS, срабатывает Широковещательное намерение с действием android.provider.Telephony.SMS_RECEIVED. Обратите внимание, что в настоящее время SDK не содержит ссылки на этот строковой литерал, поэтому в своих приложениях нужно указывать его явно.

ВНИМАНИЕ

Действие, возникающее при приеме SMS, скрыто (а значит, не поддерживается). В следующих версиях платформы оно может быть изменено. В любом случае, будьте осторожны при использовании неподдерживаемых возможностей платформы, так как они не застрахованы от изменений в будущем.

Для приложений, которые отслеживают передачу Намерений, связанных с SMS, необходимо указать полномочие RECEIVE_SMS. Сделайте это с помощью тега `<uses-permission>` в манифесте приложения, как показано в следующем фрагменте:

```
<uses-permission
  android:name="android.permission.RECEIVE_SMS"
/>
```

Широковещательное намерение с действием SMS_RECEIVED содержит информацию о входящем сообщении. Чтобы извлечь массив объектов SmsMessage, упакованных внутри дополнительного параметра Намерения, используйте ключ pdu. При этом вы получите массив значений в формате PDU (применяется для инкапсуляции SMS и его метаданных), каждое из которых представляет собой SMS. Преобразовать массив байтов формата PDU в объект SmsMessage можно с помощью метода SmsMessage.createFromPdu, как показано в листинге 12.19.

Листинг 12.19. Извлечение SMS из переданного Намерения

```
Bundle bundle = intent.getExtras();
if (bundle != null) {
  Object[] pdus = (Object[]) bundle.get("pdus");
  SmsMessage[] messages = new SmsMessage[pdus.length];
  for (int i = 0; i < pdus.length; i++)
    messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
}
```

Каждый объект SmsMessage содержит оригинальный адрес (номер телефона), временную отметку (timestamp) и тело сообщения.

В листинге 12.20 показана реализация Приемника широковещательных намерений — обработчик onReceive проверяет, начинается ли текст SMS строкой @echo, после чего пересылает этот текст обратно (на номер, с которого сообщение пришло).

Листинг 12.20. Отслеживание входящих SMS

```
public class IncomingSMSReceiver extends BroadcastReceiver {
  private static final String queryString = "@echo";
  private static final String SMS_RECEIVED =
    "android.provider.Telephony.SMS_RECEIVED";

  public void onReceive(Context _context, Intent _intent) {
    if (_intent.getAction().equals(SMS_RECEIVED)) {
      SmsManager sms = SmsManager.getDefault();

      Bundle bundle = _intent.getExtras();
      if (bundle != null) {
        Object[] pdus = (Object[]) bundle.get("pdus");
        SmsMessage[] messages = new SmsMessage[pdus.length];
        for (int i = 0; i < pdus.length; i++)
          messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);

        for (SmsMessage message : messages) {
```

```
            String msg = message.getMessageBody();
            String to = message.getOriginatingAddress();

            if (msg.toLowerCase().startsWith(queryString)) {
              String out = msg.substring(queryString.length());
              sms.sendTextMessage(to, null, out, null, null);
            }
          }
        }
      }
    }
  }
```

Чтобы отслеживать входящие сообщения, зарегистрируйте Приемник широковещательных намерений из листинга 12.20, используя Фильтр намерений для действия android.provider.Telephony.SMS_RECEIVED, как показано ниже.

Листинг 12.21. Регистрация Приемника широковещательных намерений для отслеживания входящих SMS

```
final String SMS_RECEIVED = "android.provider.Telephony.SMS_RECEIVED";
IntentFilter filter = new IntentFilter(SMS_RECEIVED);
BroadcastReceiver receiver = new IncomingSMSReceiver();
registerReceiver(receiver, filter);
```

Моделирование входящих SMS в эмуляторе

Смоделировать входящее SMS в эмуляторе можно двумя способами. Первый уже описывался в этом разделе: можно отправлять сообщение из одного эмулятора в другой, используя порт в качестве телефонного номера.

Но вы также можете использовать отладочные инструменты из Android SDK, о которых шла речь в главе 2, чтобы моделировать входящие сообщения с произвольных номеров. Данный процесс показан на рис. 12.1.

Рис. 12.1.

Работа с бинарными SMS

Бинарные сообщения принимаются точно так же, как и обычные, и могут быть извлечены по тому же принципу.

Чтобы извлечь данные, переданные внутри бинарного SMS, используйте метод getUserData, как показано в следующем фрагменте:

```
byte[] data = msg.getUserData();
```

Этот метод возвращает массив байтов — данные, содержащиеся в сообщении.

Приложение Emergency Responder

В данном примере[1] вы создадите приложения для работы с SMS, которое превратит ваш мобильный телефон в автоответчик, реагирующий на экстренные сообщения.

Если вы вдруг окажетесь поблизости от эпицентра инопланетного вторжения или случайно примете участие в восстании роботов, то сможете сделать так, чтобы ваш телефон в ответ на просьбы родственников и друзей поделиться впечатлениями о происходящем автоматически отправлял дружелюбное сообщение (или отчаянные мольбы о помощи).

Чтобы облегчить задачу потенциальным спасителям, воспользуйтесь геолокационными сервисами, передавая информацию о том, где именно вы находитесь. Устойчивость сетевой инфраструктуры SMS делает этот вид передачи данных наиболее подходящим для приложений, работа которых полностью зависит от надежности и доступности канала связи.

1. Начните с создания нового проекта EmergencyResponder, содержащего одноименную Активность.

```
package com.paad.emergencyresponder;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Locale;
import java.util.concurrent.locks.ReentrantLock;
import java.util.List;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;

import android.content.BroadcastReceiver;
import android.content.SharedPreferences;
import android.location.Address;
```

[1] Все фрагменты кода в этом примере — часть проекта Emergency Responder из главы 12, их можно загрузить с сайта Wrox.com.

```
import android.location.Geocoder;
import android.location.Location;м
import android.location.LocationManager;

import android.os.Bundle;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.ListView;

public class EmergencyResponder extends Activity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
}
```

2. Добавьте в манифест проекта полномочия, чтобы иметь возможность искать свое местоположение и отправлять/принимать SMS.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.paad.emergencyresponder">
  <application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity
      android:name=".EmergencyResponder"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <uses-permission android:name="android.permission.SEND_SMS"/>
  <uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"
  />
</manifest>
```

3. Отредактируйте ресурс с разметкой `main.xml`. Добавьте элемент ListView для отображения списка людей, которые запрашивают обновление вашего статуса, а также несколько кнопок для отправки ответных SMS. Используйте ссылки на внешние ресурсы (вы создадите их в пункте 4), чтобы указать текст для этих кнопок.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
```

```
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:id="@+id/labelRequestList" android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="These people want to know if you're ok"
    android:layout_alignParentTop="true"
  />
  <LinearLayout
    android:id="@+id/buttonLayout"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="5px"
    android:layout_alignParentBottom="true">
    <CheckBox
      android:id="@+id/checkboxSendLocation"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Include Location in Reply"/>
    <Button
      android:id="@+id/okButton"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/respondAllClearButtonText"/>
    <Button
      android:id="@+id/notOkButton"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/respondMaydayButtonText"/>
    <Button
      android:id="@+id/autoResponder"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Setup Auto Responder"/>
  </LinearLayout>
  <ListView
    android:id="@+id/myListView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_below="@id/labelRequestList"
    android:layout_above="@id/buttonLayout"/>
  </RelativeLayout>
```

4. Отредактируйте файл с внешними ресурсами `strings.xml`, добавив в него текст для каждой кнопки, а также сообщения по умолчанию, которые будут использоваться при ответе (в том числе I'm safe и I'm in danger). Вам также необходимо задать текст для распознавания входящих сообщений, на которые нужно отвечать.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Emergency Responder</string>
  <string name="respondAllClearButtonText">I am Safe and Well
```

```
  </string>
  <string name="respondMaydayButtonText">MAYDAY! MAYDAY! MAYDAY!
  </string>
  <string name="respondAllClearText">I am safe and well. Worry not!
  </string>
  <string name="respondMaydayText">Tell my mother I love her.
  </string>
  <string name="querystring">are you ok?</string>
</resources>
```

5. На данный момент графический интерфейс готов. Поэтому при запуске приложения экран должен выглядеть, как на рис. 12.2.



Рис. 12.2.

6. Создайте внутри Активности EmergencyResponder новый список со строками для хранения телефонных номеров, с которых будут приходить SMS. В методе OnCreate свяжите этот список с элементом ListView с помощью ArrayAdapter и создайте новый объект ReentrantLock, чтобы обеспечить безопасную работу со списком в поточном режиме.

   Затем получите ссылку на элемент CheckBox и добавьте объекты OnClickListener для каждой из кнопок, с помощью которых будет отправляться ответное сообщение. Первые две из них должны вызывать

метод respond, тогда как последняя будет делать вызов заглушки startAutoResponder.

```
ReentrantLock lock;
CheckBox locationCheckBox;
ArrayList<String> requesters;
ArrayAdapter<String> aa;

@Override
public void onCreate(Bundle savedInstanceState){
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  lock = new ReentrantLock();
  requesters = new ArrayList<String>();
  wireUpControls();
}

private void wireUpControls() {
  locationCheckBox = (CheckBox)findViewById(R.id.checkboxSendLocation);
  ListView myListView = (ListView)findViewById(R.id.myListView);

  int layoutID = android.R.layout.simple_list_item_1;
  aa = new ArrayAdapter<String>(this, layoutID, requesters);
  myListView.setAdapter(aa);

  Button okButton = (Button)findViewById(R.id.okButton);
  okButton.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
      respond(true, locationCheckBox.isChecked());
    }
  });

  Button notOkButton = (Button)findViewById(R.id.notOkButton);
  notOkButton.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
      respond(false, locationCheckBox.isChecked());
    }
  });

  Button autoResponderButton =
    (Button)findViewById(R.id.autoResponder);
  autoResponderButton.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
      startAutoResponder();
    }
  });
}

public void respond(boolean _ok, boolean _includeLocation) {}
private void startAutoResponder() {}
```

7. Реализуйте Приемник широковещательных намерений — он будет отслеживать входящие SMS.

7.1. Начните с воплощения нового статического строкового поля, в котором будет храниться действие для Намерения, срабатывающего при входящем сообщении.

```
public static final String SMS_RECEIVED =
  "android.provider.Telephony.SMS_RECEIVED";
```

7.2. Затем создайте новый Приемник в виде переменной внутри Активности EmergencyResponder. Он должен отслеживать входящие SMS и вызывать метод requestReceived, если обнаружит в них строку are you safe, хранящуюся во внешнем ресурсе, который был создан на шаге 4.

```
BroadcastReceiver emergencyResponseRequestReceiver =
  new BroadcastReceiver() {
    @Override
    public void onReceive(Context _context, Intent _intent) {
      if (_intent.getAction().equals(SMS_RECEIVED)) {
        String queryString = getString(R.string.querystring);

        Bundle bundle = _intent.getExtras();
        if (bundle != null) {
          Object[] pdus = (Object[]) bundle.get("pdus");
          SmsMessage[] messages = new SmsMessage[pdus.length];
          for (int i = 0; i < pdus.length; i++)
            messages[i] =
              SmsMessage.createFromPdu((byte[]) pdus[i]);

          for (SmsMessage message : messages) {
            if (message.getMessageBody().toLowerCase().contains
               (queryString))
            requestReceived(message.getOriginatingAddress());
          }
        }
      }
    }
  };

public void requestReceived(String _from) {}
```

8. Добавьте в метод onCreate из Активности EmergencyResponder регистрацию Приемника широковещательных намерений из пункта 7.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  lock = new ReentrantLock();
  requesters = new ArrayList<String>();
  wireUpControls();

  IntentFilter filter = new IntentFilter(SMS_RECEIVED);
  registerReceiver(emergencyResponseRequestReceiver, filter);
}
```

9. Перейдите к заглушке requestReceived и сделайте так, чтобы каждый телефонный номер, с которого приходило SMS, содержащее строку are you safe, добавлялся в список запросов.

```
public void requestReceived(String _from) {
  if (!requesters.contains(_from)) {
    lock.lock();
    requesters.add(_from);
    aa.notifyDataSetChanged();
    lock.unlock();
  }
}
```

10. Теперь Активность EmergencyResponder должна следить за приходом SMS и по мере поступления добавлять их в элемент ListView. Запустите программу и пошлите сообщение устройству или эмулятору, на котором эта программа функционирует. Когда сообщение придет, оно должно отобразиться в списке, как показано на рис. 12.3.



Рис. 12.3.

11. Отредактируйте Активность таким образом, чтобы пользователь мог отвечать на эти сообщения.

Начните с реализации метода-заглушки respond, который вы создали в пункте 6. Он должен перебрать все номера в списке и отослать по каждому из них SMS, текст которого будет основываться на строках, заданных вами в виде ресурсов в пункте 4. Сообщения будут передаваться с помощью перегруженного метода respond, работу над которым вы завершите на следующем шаге.

```
public void respond(boolean _ok, boolean _includeLocation) {
  String okString = getString(R.string.respondAllClearText);
  String notOkString = getString(R.string.respondMaydayText);

  String outString = _ok ? okString : notOkString;

  ArrayList<String> requestersCopy =
    (ArrayList<String>)requesters.clone();

  for (String to : requestersCopy)
    respond(to, outString, _includeLocation);
}

private void respond(String _to, String _response,
                     boolean _includeLocation) {}
```

12. Измените метод respond, который отвечает за отправку каждого ответного SMS.

Прежде чем слать сообщение, удалите всех потенциальных получателей из списка. Если ответ должен включать данные о текущем местоположении, получите их с помощью объекта LocationManager, затем отправьте второе сообщение, содержащие ваши долготу/широту и адрес из геокодировщика.

```
public void respond(String _to, String _response,
                    boolean _includeLocation) {
  // Удалите адресата из списка людей,
  // которым мы должны отвечать.
  lock.lock();
  requesters.remove(_to);
  aa.notifyDataSetChanged();
  lock.unlock();

  SmsManager sms = SmsManager.getDefault();

  // Отправьте сообщение
  sms.sendTextMessage(_to, null, _response, null, null);

  StringBuilder sb = new StringBuilder();

  // Если это необходимо, получите текущее
  // местоположение и отправьте его по SMS.
  if (_includeLocation) {
    String ls = Context.LOCATION_SERVICE;
    LocationManager lm = (LocationManager)getSystemService(ls);
```

```
      Location l =
        lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);

      sb.append("I'm @:\n");
      sb.append(l.toString() + "\n");

      List<Address> addresses;
      Geocoder g = new Geocoder(getApplicationContext(),
                                Locale.getDefault());
      try {
        addresses = g.getFromLocation(l.getLatitude(),
                                      l.getLongitude(), 1);
        if (addresses != null) {
          Address currentAddress = addresses.get(0);
          if (currentAddress.getMaxAddressLineIndex() > 0) {
            for (int i = 0;
                 i < currentAddress.getMaxAddressLineIndex();
                 i++)
            {
              sb.append(currentAddress.getAddressLine(i));
              sb.append("\n");
            }
            }
            else {
              if (currentAddress.getPostalCode() != null)
                sb.append(currentAddress.getPostalCode());
          }
        }
      } catch (IOException e) {}

      ArrayList<String> locationMsgs =
        sms.divideMessage(sb.toString());
      for (String locationMsg : locationMsgs)
        sms.sendTextMessage(_to, null, locationMsg, null, null);
    }
  }
```

13. В экстренных ситуациях чрезвычайно важно, чтобы сообщения доходили до адресата. Улучшите надежность своего приложения, добавив в него возможность повторной отправки. Отслеживая успешность доставки SMS, вы можете отослать его второй раз, если в первый возникли какие-то проблемы.

    13.1. Начните с создания нового статического строкового поля в Активности EmergencyResponder. Оно будет использоваться для определения локального действия SMS_SENT.

```
public static final String SENT_SMS =
  "com.paad.emergencyresponder.SMS_SENT";
```

    13.2. Отредактируйте метод respond, добавив в него новое Намерение PendingIntent, которое будет передавать действие, созданное в предыдущем пункте (сигнализируя о том, что отправка сообще-

ния завершена). Намерение должно включать номер получателя в виде дополнительного параметра.

```
public void respond(String _to, String _response,
                    boolean _includeLocation) {
  // Удалите адресата из списка людей,
  // которым мы должны отвечать.
  lock.lock();
  requesters.remove(_to);
  aa.notifyDataSetChanged();
  lock.unlock();

  SmsManager sms = SmsManager.getDefault();

  Intent intent = new Intent(SENT_SMS);
  intent.putExtra("recipient", _to);

  PendingIntent sent =
    PendingIntent.getBroadcast(getApplicationContext(),
                               0, intent, 0);

  // Отправьте сообщение
  sms.sendTextMessage(_to, null, _response, sent, null);

  StringBuilder sb = new StringBuilder();

  if (_includeLocation) {
    [ . . . ранее написанный код для поиска местоположения . . . ]
    ArrayList<String> locationMsgs =
      sms.divideMessage(sb.toString());
    for (String locationMsg : locationMsgs)
      sms.sendTextMessage(_to, null, locationMsg, sentIntent, null);
  }
}
```

13.3. Реализуйте новый абстрактный класс BroadcastReceiver для отслеживания этого Намерения. Переопределите его обработчик onReceive, чтобы подтверждать успешность доставки SMS. Если сообщение не было доставлено, адресата необходимо вернуть обратно в список.

```
private BroadcastReceiver attemptedDeliveryReceiver = new
BroadcastReceiver() {
  @Override
  public void onReceive(Context _context, Intent _intent) {
    if (_intent.getAction().equals(SENT_SMS)) {
      if (getResultCode() != Activity.RESULT_OK) {
        String recipient = _intent.getStringExtra("recipient");
        requestReceived(recipient);
      }
    }
  }
};
```

13.4. В завершение зарегистрируйте новый Приемник широковещательных намерений в методе onCreate, принадлежащем Активности EmergencyResponder.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  lock = new ReentrantLock();
  requesters = new ArrayList<String>();
  wireUpControls();

  IntentFilter filter = new IntentFilter(SMS_RECEIVED);
  registerReceiver(emergencyResponseRequestReceiver, filter);

  IntentFilter attemptedDeliveryfilter = new IntentFilter(SENT_SMS);
  registerReceiver(attemptedDeliveryReceiver,
                   attemptedDeliveryfilter);
}
```

Данный пример сознательно упрощен, чтобы сосредоточить ваше внимание на работе с SMS. Внимательные читатели должны были заметить по крайней мере две вещи, которые могут быть улучшены.

- Регистрацию Приемника широковещательных намерений, созданного и зарегистрированного в пунктах 7 и 8, предпочтительнее проводить внутри манифеста, чтобы приложение могло отвечать на входящие SMS, даже если оно не запущено.
- Обработка входящих SMS, выполняющаяся внутри Приемника (пункты 7 и 9), должна быть перенесена в Сервис и работать в фоновом потоке. То же самое следует сделать с отправкой ответных сообщений, описанной в пункте 13.

Предлагаем реализовать данные улучшения самостоятельно, пользуясь навыками, приобретенными в главе 9.

Автоматизация приложения Emergency Responder

В следующем примере[1] вы добавите функциональность к кнопке Setup Auto Responder, созданной ранее. С ее помощью ваше приложение сможет автоматически отвечать на полученные сообщения.

1. Начните с создания нового ресурса разметки autoresponder.xml, в котором будет описано окно с настройками для автоматических ответов. Добавьте туда элементы EditText для ввода текста сообщения, Spinner для выбора времени, в течение которого будет отослан

[1] Все фрагменты кода в этом примере — часть проекта Emergency Responder 2 из главы 12, их можно загрузить с сайта Wrox.com.

автоматический ответ, и CheckBox, чтобы пользователь мог выбрать, нужно добавлять в ответ свое местоположение или нет.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Respond With"/>
  <EditText
    android:id="@+id/responseText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
  <CheckBox
    android:id="@+id/checkboxLocation" android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Transmit Location"/>
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Auto Respond For"/>
  <Spinner
    android:id="@+id/spinnerRespondFor"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:drawSelectorOnTop="true"/>
  <LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <Button
      android:id="@+id/okButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Enable"/>
    <Button
      android:id="@+id/cancelButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Disable"/>
  </LinearLayout>
</LinearLayout>
```

2. Отредактируйте ресурс `string.xml`, определив имя Активности как SharedPreference и строки для каждого из ее ключей.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Emergency Responder</string>
  <string name="respondAllClearButtonText">I am Safe and Well
```

```
  </string>
  <string name="respondMaydayButtonText">MAYDAY! MAYDAY! MAYDAY!
  </string>
  <string name="respondAllClearText">I am safe and well. Worry not!
  </string>
  <string name="respondMaydayText">Tell my mother I love her.
  </string>
  <string name="querystring">"are you ok?"</string>

  <string
    name="user_preferences">com.paad.emergencyresponder.preferences
  </string>
  <string name="includeLocationPref">PREF_INCLUDE_LOC</string>
  <string name="responseTextPref">PREF_RESPONSE_TEXT</string>
  <string name="autoRespondPref">PREF_AUTO_RESPOND</string>
  <string name="respondForPref">PREF_RESPOND_FOR</string>
</resources>
```

3. Создайте новый ресурс `arrays.xml`, который будет содержать массив для заполнения данными элемента Spinner.

```
<resources>
  <string-array name="respondForDisplayItems">
    <item>- Disabled -</item>
    <item>Next 5 minutes</item>
    <item>Next 15 minutes</item>
    <item>Next 30 minutes</item>
    <item>Next hour</item>
    <item>Next 2 hours</item>
    <item>Next 8 hours</item>
  </string-array>

  <array name="respondForValues">
    <item>0</item>
    <item>5</item>
    <item>15</item>
    <item>30</item>
    <item>60</item>
    <item>120</item>
    <item>480</item>
  </array>
</resources>
```

4. Создайте новую Активность AutoResponder и привяжите ее к разметке из пункта 1.

```
package com.paad.emergencyresponder;

import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.res.Resources;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
```

```
import android.content.BroadcastReceiver;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.Spinner;

public class AutoResponder extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.autoresponder);
  }
}
```

5. Отредактируйте заглушку onCreate, чтобы получить ссылки на каждое Представление внутри разметки и заполнить элемент Spinner данными из массива, заданного в пункте 3. Создайте два новых метода-заглушки — savePreferences и updateUIFromPreferences. С помощью первого настройки будут сохраняться в объект SharedPreferences, с помощью второго — применяться для текущего пользовательского интерфейса.

```
Spinner respondForSpinner;
CheckBox locationCheckbox;
EditText responseTextBox;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.autoresponder);
```

5.1. Получите ссылки на каждое Представление.

```
respondForSpinner = (Spinner)findViewById(R.id.spinnerRespondFor);
locationCheckbox = (CheckBox)findViewById(R.id.checkboxLocation);
responseTextBox = (EditText)findViewById(R.id.responseText);
```

5.2. Заполните данными элемент Spinner, чтобы пользователи могли выбирать время, в течение которого отсылается автоматический ответ.

```
ArrayAdapter<CharSequence> adapter =
  ArrayAdapter.createFromResource(this,
    R.array.respondForDisplayItems,
    android.R.layout.simple_spinner_item);

adapter.setDropDownViewResource(
  android.R.layout.simple_spinner_dropdown_item);
respondForSpinner.setAdapter(adapter);
```

5.3. Теперь добавьте обработку нажатий для кнопок OK и Cancel, чтобы пользователи могли сохранять или отменять изменения в настройках.

```
Button okButton = (Button) findViewById(R.id.okButton);
okButton.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    savePreferences();
    setResult(RESULT_OK, null);
    finish();
  }
});

Button cancelButton = (Button) findViewById(R.id.cancelButton);
cancelButton.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    respondForSpinner.setSelection(-1);
    savePreferences();
    setResult(RESULT_CANCELED, null);
    finish();
  }
});
```

5.4. В завершение сделайте так, чтобы состояние пользовательского интерфейса Активности при запуске соответствовало текущим настройкам.

```
// Загрузите сохраненные настройки и обновите пользовательский интерфейс
updateUIFromPreferences();
```

5.5. Закончив с методом onCreate, следует добавить две заглушки — updateUIFromPreferences и savePreferences.

```
}

private void updateUIFromPreferences() {}
private void savePreferences() {}
```

6. Наполните их кодом. Начните с метода updateUIFromPreferences. Он должен считывать текущие настройки Активности AutoResponder и применять их к пользовательскому интерфейсу.

```
private void updateUIFromPreferences() {
  // Получите сохраненные настройки
  String preferenceName = getString(R.string.user_preferences);
  SharedPreferences sp = getSharedPreferences(preferenceName, 0);

  String autoResponsePref = getString(R.string.autoRespondPref);
  String responseTextPref = getString(R.string.responseTextPref);
  String autoLocPref = getString(R.string.includeLocationPref);
  String respondForPref = getString(R.string.respondForPref);

  boolean autoRespond = sp.getBoolean(autoResponsePref, false);
  String respondText = sp.getString(responseTextPref, "");
  boolean includeLoc = sp.getBoolean(includeLocPref, false);
```

```
  int respondForIndex = sp.getInt(respondForPref, 0);

  // Примените сохраненные настройки к пользовательскому интерфейсу
  if (autoRespond)
    respondForSpinner.setSelection(respondForIndex);
  else
    respondForSpinner.setSelection(0);

  locationCheckbox.setChecked(includeLoc);
  responseTextBox.setText(respondText);
}
```

7. Заполните кодом заглушку savePreferences, чтобы сохранять текущее состояние пользовательского интерфейса в файл Общих настроек.

```
private void savePreferences() {
  // Получите текущее состояние пользовательского интерфейса
  boolean autoRespond =
    respondForSpinner.getSelectedItemPosition() > 0;
  int respondForIndex = respondForSpinner.getSelectedItemPosition();
  boolean includeLoc = locationCheckbox.isChecked();
  String respondText = responseTextBox.getText().toString();

  // Сохраните его в файл Общих Настроек
  String preferenceName = getString(R.string.user_preferences);
  SharedPreferences sp = getSharedPreferences(preferenceName, 0);

  Editor editor = sp.edit();
  editor.putBoolean(getString(R.string.autoRespondPref),
                    autoRespond);
  editor.putString(getString(R.string.responseTextPref),
                             respondText);
  editor.putBoolean(getString(R.string.includeLocationPref),
                              includeLoc );
  editor.putInt(getString(R.string.respondForPref),respondForIndex);
  editor.commit();

  // Отмените автоматические ответы с помощью метода setAlarm
  setAlarm(respondForIndex);
}

private void setAlarm(int respondForIndex) {}
```

8. Заглушка setAlarm из предыдущего пункта нужна для создания новой Сигнализации. Она должна активизировать Намерение, которое отключит автоматические ответы.

   Вам необходимо создать новый объект BroadcastReceiver, который будет отслеживать Намерения, посланные Сигнализацией, прежде чем отключить автоответ.

   8.1. Начните с создания строкового поля для действия, которое будет представлять Намерение для Сигнализации.

```
public static final String alarmAction =
  "com.paad.emergencyresponder.AUTO_RESPONSE_EXPIRED";
```

8.2. Создайте новый экземпляр BroadcastReceiver, который будет отслеживать Намерение, содержащее действие из пункта 8.1. Получив Намерение, он должен соответствующим образом изменить настройки автоответчика.

```
private BroadcastReceiver stopAutoResponderReceiver = new
BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals(alarmAction)) {
      String preferenceName = getString(R.string.user_preferences);
      SharedPreferences sp = getSharedPreferences(preferenceName,0);

      Editor editor = sp.edit();
      editor.putBoolean(getString(R.string.autoRespondPref), false);
      editor.commit();
    }
  }
};
```

8.3. Закончите метод setAlarm. Он должен отменять Сигнализацию, если автоматический ответ уже выключен; в противном случае нужно будет задать для Сигнализации временной интервал, по истечении которого она должна сработать.

```
PendingIntent intentToFire;

private void setAlarm(int respondForIndex) {
  // Создайте Сигнализацию и зарегистрируйте
  // для нее Приемник широковещательных намерений.

  AlarmManager alarms =
    (AlarmManager)getSystemService(ALARM_SERVICE);

  if (intentToFire == null) {
    Intent intent = new Intent(alarmAction);
    intentToFire =
      PendingIntent.getBroadcast(getApplicationContext(),
                                 0,intent,0);

    IntentFilter filter = new IntentFilter(alarmAction);

    registerReceiver(stopAutoResponderReceiver, filter);
  }

  if (respondForIndex < 1)
    // Если автоответчик отключен, отмените Сигнализацию.
    alarms.cancel(intentToFire);
  else {
    // В ином случае получите временной интервал,
    // выбранный в настройках, и сделайте так, чтобы
    // Сигнализация сработала по его истечении.
    Resources r = getResources();
```

```
    int[] respondForValues =
      r.getIntArray(R.array.respondForValues);
    int respondFor = respondForValues [respondForIndex];

    long t = System.currentTimeMillis();
    t = t + respondFor*1000*60;

    // Установите Сигнализацию.
    alarms.set(AlarmManager.RTC_WAKEUP, t, intentToFire);
  }
}
```

9. На этом написание Активности AutoResponder завершается. Но прежде чем использовать, необходимо добавить ее в манифест приложения.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.paad.emergencyresponder">
  <application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity
      android:name=".EmergencyResponder"
      android:label="@string/app_name">
      <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
      android:name=".AutoResponder"
      android:label="Auto Responder Setup"/>
  </application>
  <uses-permission android:name="android.permission.ACCESS_GPS"/>
  <uses-permission android:name="android.permission.ACCESS_LOCATION"/>
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <uses-permission android:name="android.permission.SEND_SMS"/>
</manifest>
```

10. Для того чтобы включить автоматический ответ, вернитесь к Активности EmergencyResponder и отредактируйте заглушку startAutoResponder, созданную в предыдущем примере. Она должна открывать Активность AutoResponder, написание которой вы только что закончили.

```
private void startAutoResponder() {
  startActivityForResult(new Intent(EmergencyResponder.this,
                        AutoResponder.class), 0);
}
```

11. Теперь, запустив проект, вы сможете открыть окно с настройками и изменить поведение автоматического ответа. Выглядеть это должно, как на рис. 12.4.

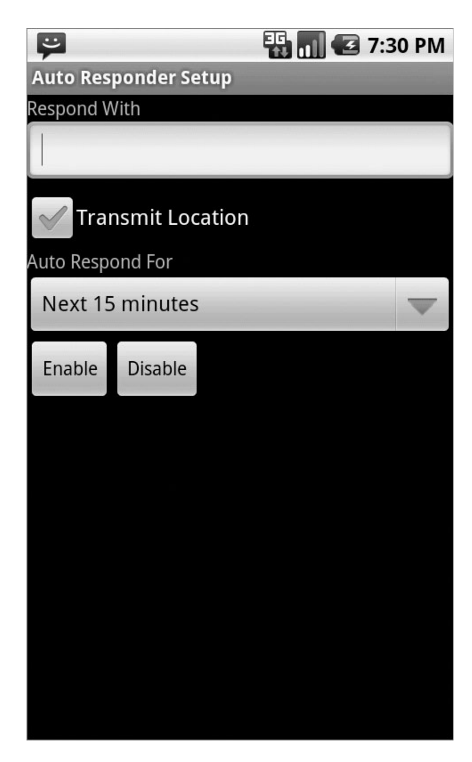


Рис. 12.4.

12. И последний шаг: отредактируйте метод requestReceived из Активности EmergencyResponder, чтобы проверять, включена ли функция автоматического ответа.

    В случае положительного ответа должен автоматически запуститься метод respond с использованием настроек для сообщения и местоположения, заданных в Общих настройках приложения.

```
public void requestReceived(String _from) {
  if (!requesters.contains(_from)) {
    lock.lock();
    requesters.add(_from);
    aa.notifyDataSetChanged();
    lock.unlock();

    // Проверьте настройки автоответчика
    String preferenceName = getString(R.string.user_preferences);
    SharedPreferences prefs = getSharedPreferences(preferenceName,
                                                   0);
    String autoRespondPref = getString(R.string.autoRespondPref)
    boolean autoRespond = prefs.getBoolean(autoRespondPref, false);

    if (autoRespond) {
      String responseTextPref =
        getString(R.string.responseTextPref);
```

```
        String includeLocationPref =
          getString(R.string.includeLocationPref);

        String respondText = prefs.getString(responseTextPref, "");
        boolean includeLoc = prefs.getBoolean(includeLocationPref,
                                              false);

        respond(_from, respondText, includeLoc);
      }
    }
  }
```

Теперь у вас есть полнофункциональное интерактивное приложение, способное автоматически отвечать на экстренные сообщения.

Резюме

Телефонный стек — одна из фундаментальных технологий на мобильных телефонах. И хотя не все устройства под управлением Android должны ее поддерживать, она стала основой для универсальной платформы, обеспечивающей связь между людьми.

Используя API для телефонии, вы научились делать звонки вручную и через стандартное приложение. Вы также узнали, как считывать и отслеживать состояния телефона, сети (в том числе сети для передачи данных) и SIM-карты.

Android позволяет использовать SMS для создания приложений, обеспечивающих обмен данными между устройствами и передачу сообщений между пользователями.

Вы также узнали, как использовать Намерения, чтобы с помощью системных приложений, изначально установленных на телефоне, отправлять SMS и MMS от своего имени.

В главе 13 рассмотрены другие коммуникационные технологии, доступные на устройствах. Вы изучите управление сетевыми подключениями на примере Wi-Fi и исследуете возможности программных интерфейсов для работы с Bluetooth.

Глава 13
BLUETOOTH, WI-FI, СЕТЬ

Содержание главы

- Управление Bluetooth- устройствами.
- Обнаружение Bluetooth-устройств.
- Управление режимом обнаружения.
- Взаимодействие через Bluetooth.
- Отслеживание подключения к Интернету.
- Соблюдение пользовательских настроек при передаче данных в фоновом режиме.
- Отслеживание информации о Wi-Fi и других сетях.
- Изменение конфигурации Wi-Fi и других сетей.
- Поиск точек доступа к сетям Wi-Fi.

В этой главе вы продолжите изучать низкоуровневые программные интерфейсы в Android, исследуя пакеты для работы с Bluetooth, Wi-Fi и другими сетями.

Android предоставляет API для управления и слежения за настройками вашего Bluetooth-устройства, чтобы контролировать его доступность. Вы сможете искать Bluetooth-устройства по соседству и использовать этот протокол в своих приложениях в качестве децентрализованного (p2p) слоя для передачи данных.

Вам также доступен полноценный пакет для работы с Wi-Fi и другими сетями. Используя эти API, можно искать точки доступа, создавать и изменять настройки Wi-Fi, отслеживать подключения к Интернету, а также следить за свойствами и настройками этих подключений и управлять ими.

Использование Bluetooth

В этом разделе вы научитесь работать с локальным Bluetooth-адаптером и налаживать с его помощью связь с телефонами, находящимися поблизости.

Применяя Bluetooth, можно находить другие устройства, расположенные в пределах досягаемости, и подключаться к ним. Используя подключение с помощью Bluetooth-сокетов, вы получаете возможность передавать и принимать потоки данных между устройствами в контексте своих приложений.

ВНИМАНИЕ

Библиотеки для работы с Bluetooth стали доступны в Android только с версии 2.0 (SDK API level 5). Важно также помнить, что аппаратную поддержку Bluetooth имеют не только устройства под управлением Android.

Bluetooth — это сетевой протокол, созданный для связи в условиях низкой пропускной способности и ограниченного радиуса действия. В Android 2.1 поддерживаются исключительно зашифрованные соединения, поэтому вы можете устанавливать связь только между спаренными устройствами. Для работы с Bluetooth-устройствами и соединениями в Android существует несколько классов.

- BluetoothAdapter. Представляет собой локальный Bluetooth-адаптер, то есть устройство, на котором работает ваше приложение.
- BluetoothDevice. Этот класс нужен для удаленных устройств, к которым вы хотите подключиться.
- BluetoothSocket. Чтобы получить экземпляр данного класса, вызовите метод createRfcommSocketToServiceRecord из объекта BluetoothDevice. Это позволит создать соединение с удаленным устройством.
- BluetoothServerSocket. Экземпляр этого класса (созданный с помощью метода listenUsingRfcommWithServiceRecord) для объекта BluetoothAdapter дает возможность отслеживать входящие сетевые запросы, поступающие от удаленных устройств.

Доступ к локальному Bluetooth-устройству

Локальное Bluetooth-устройство управляется классом BluetoothAdapter.

Чтобы получить доступ к Bluetooth-адаптеру, используемому по умолчанию, вызовите метод getDefaultAdapter, как показано в листинге 13.1. Телефоны под управлением Android могут иметь несколько Bluetooth-устройств, но на сегодняшний день доступ можно получить только к тому, которое является Адаптером по умолчанию.

Листинг 13.1. Доступ к Bluetooth-адаптеру по умолчанию

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
```

Чтобы получить какие-либо свойства локального адаптера, начать сканирование или найти привязанные устройства, необходимо добавить в манифест своего приложения полномочие BLUETOOTH. Для изменения

любых свойств локального Bluetooth-адаптера также потребуется полномочие BLUETOOTH_ADMIN.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Управление свойствами и состоянием Bluetooth-адаптера

Класс BluetoothAdapter содержит методы для считывания и изменения свойств локального Bluetooth-устройства.

ПРИМЕЧАНИЕ

Свойства Bluetooth-адаптера могут быть прочитаны и изменены только в том случае, если он включен (то есть если устройство имеет статус включенного). Если же устройство выключено, любой из этих методов вернет значение null.

Если Bluetooth-адаптер включен и в манифесте приложения содержится полномочие BLUETOOTH, вы можете получить доступ к «дружественному имени» устройства (произвольная строка, которую пользователь может задать и затем использовать для идентификации конкретного устройства), а также к аппаратному адресу, как показано в листинге 13.2.

Прежде чем начать работу с этими свойствами, с помощью метода isEnabled удостоверитесь, что устройство включено.

Листинг 13.2. Считывание свойств Bluetooth-адаптера

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();

String toastText;
if (bluetooth.isEnabled()) {
  String address = bluetooth.getAddress();
  String name = bluetooth.getName();
  toastText = name + " : " + address;
}
else
  toastText = "Bluetooth is not enabled";

Toast.makeText(this, toastText, Toast.LENGTH_LONG).show();
```

Если есть полномочие BLUETOOTH_ADMIN, вы имеете возможность изменять имя адаптера, используя метод setName:

```
bluetooth.setName("Blackfang");
```

Для более подробной информации о состоянии Bluetooth-адаптера воспользуйтесь методом getState, который вернет одну из нижеследующих констант, принадлежащих классу BluetoothAdapter:

- STATE_TURNING_ON;
- STATE_ON;

- STATE_TURNING_OFF;
- STATE_OFF.

Изначально Bluetooth-адаптер выключен (для экономии заряда батареи и из соображений безопасности большинство пользователей включают его, когда он нужен).

Чтобы активизировать Bluetooth-адаптер, запустите системное диалоговое окно, используя статическую константу из класса BluetoothAdapter в качестве параметра для метода startActivityForResult:

```
String enableBT = BluetoothAdapter.ACTION_REQUEST_ENABLE;
startActivityForResult(new Intent(enableBT), 0);
```

Результат показан на рис. 13.1. Система сообщает пользователю, что приложение требует включить Bluetooth-адаптер, и запрашивает подтверждение. Если подтверждение будет получено, адаптер включится (или произойдет ошибка), диалоговое окно закроется, а на передний план выйдет Активность, из которой оно было вызвано. Если пользователь запретит включение, диалоговое окно немедленно закроется. Используйте код результата, получаемый обработчиком onActivityResult, чтобы убедиться в успешности этих действий.

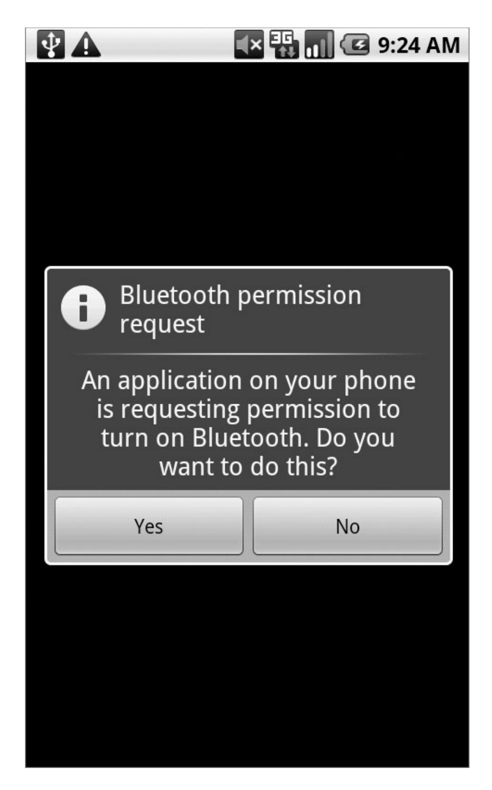


Рис. 13.1.

ВНИМАНИЕ

Если в манифесте приложения содержится полномочие BLUETOOTH_ADMIN, вы можете включать и выключать Bluetooth-адаптер с помощью методов enable и disable.

Имейте в виду, что такой подход применяется только при крайней необходимости и пользователь всегда должен быть оповещен о смене состояния адаптера. В большинстве случаев лучше использовать механизм, описанный ранее.

Включение и выключение Bluetooth-адаптера занимает некоторое время и происходит в асинхронном режиме. Вместо того чтобы постоянно опрашивать устройство, ваше приложение должно зарегистрировать Приемник широковещательных намерений, который будет следить за действием ACTION_STATE_CHANGED. Транслируемое Намерение будет содержать два дополнительных параметра — EXTRA_STATE и EXTRA_PREVIOUS_STATE, указывающих на текущее и предыдущее состояния адаптера соответственно.

В листинге 13.3 демонстрируется Намерение, с помощью которого пользователю предлагается активизировать Bluetooth, а также объект BroadcastReceiver, отслеживающий изменения в состоянии адаптера.

Листинг 13.3. Включение Bluetooth и отслеживание состояния адаптера

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();

BroadcastReceiver bluetoothState = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String prevStateExtra = BluetoothAdapter.EXTRA_PREVIOUS_STATE;
    String stateExtra = BluetoothAdapter.EXTRA_STATE;
    int state = intent.getIntExtra(stateExtra, -1);
    int previousState = intent.getIntExtra(prevStateExtra, -1);

    String tt = "";
    switch (state) {
      case (BluetoothAdapter.STATE_TURNING_ON) : {
        tt = "Bluetooth turning on"; break;
      }
      case (BluetoothAdapter.STATE_ON) : {
        tt = "Bluetooth on";
        unregisterReceiver(this);
        break;
      }
      case (BluetoothAdapter.STATE_TURNING_OFF) : {
        tt = "Bluetooth turning off"; break;
      }
      case (BluetoothAdapter.STATE_OFF) : {
        tt = "Bluetooth off"; break;
      }
```

```
      default: break;
    }

    Toast.makeText(this, tt, Toast.LENGTH_LONG).show();
  }
};

if (!bluetooth.isEnabled()) {
  String actionStateChanged = BluetoothAdapter.ACTION_STATE_CHANGED;
  String actionRequestEnable = BluetoothAdapter.ACTION_REQUEST_ENABLE;
  registerReceiver(bluetoothState,
                   new IntentFilter(actionStateChanged));
  startActivityForResult(new Intent(actionRequestEnable), 0);
}
```

Доступность для обнаружения и сканирование удаленных устройств

Процесс, когда два устройства ищут друг друга, чтобы установить соединение, называется обнаружением. Прежде чем вы сможете пользоваться объектом BluetoothSocket, локальный Bluetooth-адаптер должен связаться с удаленным устройством. Но для этого нужно, чтобы оба устройства обладали возможностью «увидеть» друг друга.

ПРИМЕЧАНИЕ

Хотя протокол Bluetooth поддерживает динамические (ad-hoc) соединения для передачи данных, этот механизм пока что недоступен в Android. В настоящее время соединение поддерживается только между связанными устройствами.

Управление обнаруживаемостью устройства

Чтобы ваш локальный Bluetooth-адаптер во время сканирования мог быть обнаружен удаленным устройством на базе Android, необходимо сделать его доступным для обнаружения.

Обнаруживаемость Bluetooth-адаптера зависит от установленного режима для сканирования, который вы можете узнать, вызвав метод getScanMode из объекта BluetoothAdapter. К вам вернется одна из констант.

- SCAN_MODE_CONNECTABLE_DISCOVERABLE. Возможны оба режима сканирования — inquiry scan и page scan. Это значит, что адаптер доступен при сканировании любым Bluetooth-устройством.
- SCAN_MODE_CONNECTABLE. Сканирование возможно только в режиме page scan. Это значит, что локальный адаптер может быть обнаружен только тем устройством, которое уже ранее было к нему подключено и привязано. Новые устройства ничего не обнаружат.

- SCAN_MODE_NONE. Локальный адаптер не может быть обнаружен никаким удаленным устройством.

Для обеспечения конфиденциальности устройства на базе Android по умолчанию имеют режим SCAN_MODE_NONE. Чтобы это изменить, вам нужно получить явное подтверждение от пользователя. Сделать это можно с помощью запуска новой Активности, используя действие ACTION_REQUEST_DISCOVERABLE:

```
String aDiscoverable = BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE;
startActivityForResult(new Intent(aDiscoverable),
                       DISCOVERY_REQUEST);
```

По умолчанию установленный режим проработает две минуты. Вы можете изменить эту цифру, добавив к Намерению дополнительный параметр EXTRA_DISCOVERABLE_DURATION, указав время (в секундах), на протяжении которого должен сохраняться данный режим.

При передаче этого Намерения на экране появится диалоговое окно (рис. 13.2), в котором пользователю будет предложено сделать локальный Bluetooth-адаптер видимым для других устройств на протяжении указанного промежутка времени.

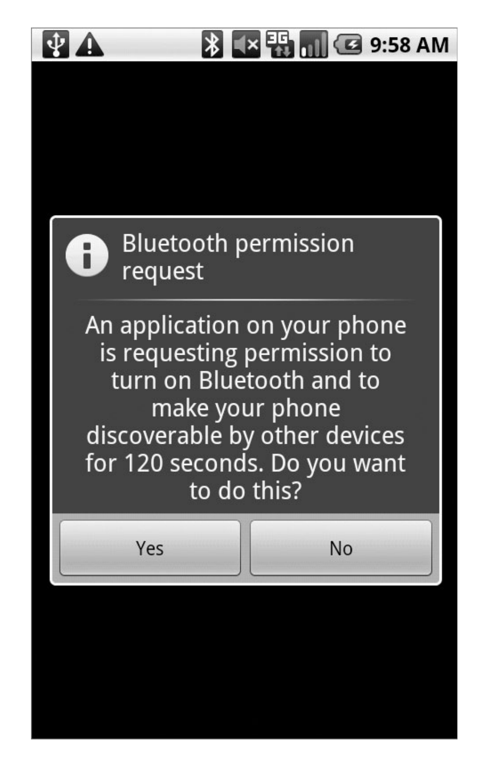


Рис. 13.2.

Чтобы узнать о выборе пользователя, переопределите обработчик onActivityResult, как показано в листинге 13.4. Возвращаемый параметр resultCode в случае положительного ответа будет содержать временной интервал. Если пользователь запретил делать Bluetooth-адаптер доступным для сканирования, resultCode будет иметь отрицательное значение.

Листинг 13.4. Отслеживание режимов доступности

```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
  if (requestCode == DISCOVERY_REQUEST) {
    boolean isDiscoverable = resultCode > 0;
    int discoverableDuration = resultCode;
  }
}
```

В качестве альтернативного подхода можно отслеживать изменения в состоянии доступности адаптера, получая действие ACTION_SCAN_MODE_CHANGED, как показано в листинге 13.5. Широковещательное намерение в виде дополнительных параметров содержит текущий и предыдущий режимы сканирования.

Листинг 13.5. Альтернативное отслеживание режимов доступности

```
registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String prevScanMode = BluetoothAdapter.EXTRA_PREVIOUS_SCAN_MODE;
    String scanMode = BluetoothAdapter.EXTRA_SCAN_MODE;
    int scanMode = intent.getIntExtra(scanMode, -1);

    int prevMode = intent.getIntExtra(prevScanMode, -1);
  }
};
new IntentFilter(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED));
```

Поиск удаленных устройств

В этом разделе вы узнаете, как инициировать процесс сканирования на своем локальном Bluetooth-адаптере, чтобы найти доступные поблизости устройства.

ПРИМЕЧАНИЕ

Процесс обнаружения может занять некоторое время (до 12 секунд). При этом производительность адаптера заметно снизится. Используйте механизм, описанный в этом разделе, чтобы проверять и отслеживать статус доступности Bluetooth-адаптера, избегая операций, требующих высокой пропускной способности (в том числе подключение к удаленному устройству), пока выполняется процесс обнаружения.

Узнать о том, занят ли Bluetooth-адаптер процессом сканирования, вы можете с помощью метода isDiscovering.

Чтобы начать обнаружение, вызовите метод startDiscovery из объекта BluetoothAdapter; чтобы его отменить, воспользуйтесь методом cancelDiscovery.

```
bluetooth.startDiscovery();
bluetooth.cancelDiscovery();
```

Процесс обнаружения идет в асинхронном режиме. Android использует Широковещательные намерения, чтобы уведомлять ваше приложение о начале и конце сканирования, а также об устройствах, которые были обнаружены.

Вы можете отслеживать изменения в ходе обнаружения, создав Приемник широковещательных намерений, настроенный на действия ACTION_DISCOVERY_STARTED и ACTION_DISCOVERY_FINISHED, как показано в листинге 13.6.

Листинг 13.6. Отслеживание процесса обнаружения

```
BroadcastReceiver discoveryMonitor = new BroadcastReceiver() {

  String dStarted = BluetoothAdapter.ACTION_DISCOVERY_STARTED;
  String dFinished = BluetoothAdapter.ACTION_DISCOVERY_FINISHED;

  @Override
  public void onReceive(Context context, Intent intent) {
    if (dStarted.equals(intent.getAction())) {
      // Процесс обнаружения начался.
      Toast.makeText(getApplicationContext(),
      "Discovery Started...", Toast.LENGTH_SHORT).show();
    }
    else if (dFinished.equals(intent.getAction())) {
      // Процесс обнаружения завершился.
      Toast.makeText(getApplicationContext(),
      "Discovery Completed...", Toast.LENGTH_SHORT).show();
    }
  }
};
registerReceiver(discoveryMonitor,
                 new IntentFilter(dStarted));
registerReceiver(discoveryMonitor,
                 new IntentFilter(dFinished));
```

Обнаруженные Bluetooth-устройства возвращаются через Широковещательные намерения с действием ACTION_FOUND.

Как показано в листинге 13.7, каждое такое Намерение содержит имя удаленного устройства в качестве дополнительного параметра с ключом BluetoothDevice.EXTRA_NAME, а также неизменяемый объект BluetoothDevice (реализующий интерфейс Parcelable), хранящийся в дополнительном параметре с ключом BluetoothDevice.EXTRA_DEVICE.

Листинг 13.7. Обнаружение удаленных Bluetooth-устройств

```
BroadcastReceiver discoveryResult = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String remoteDeviceName =
      intent.getStringExtra(BluetoothDevice.EXTRA_NAME);

    BluetoothDevice remoteDevice;
    remoteDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

    Toast.makeText(getApplicationContext(),
                   "Discovered: " + remoteDeviceName,
                   Toast.LENGTH_SHORT).show();

    // TODO Сделать что-нибудь с объектом BluetoothDevice.
  }
};
registerReceiver(discoveryResult,
                 new IntentFilter(BluetoothDevice.ACTION_FOUND));

if (!bluetooth.isDiscovering())
  bluetooth.startDiscovery();
```

Объект BluetoothDevice, возвращенный через Намерение, — это обнаруженное Bluetooth-устройство. В следующих разделах он будет использоваться для создания соединения, привязки и в конечном счете передачи данных между локальным адаптером и удаленным устройством.

Использование Bluetooth для связи

API для установления связи через Bluetooth считается оберткой вокруг протокола RFCOMM (Radio Frequency Communication). RFCOMM, в свою очередь, поддерживает стандарт передачи данных RS232 поверх протокола L2CAP (Logical Link Control and Adaptation Protocol).

На практике все эти непонятные аббревиатуры можно назвать механизмом для открытия сетевых сокетов между двумя связанными Bluetooth-устройствами.

ПРИМЕЧАНИЕ

Прежде чем ваше приложение сможет устанавливать связь между устройствами, их адаптеры должны быть спарены (связаны). На момент написания этой книги (Android API level 7) не существовало способа вручную связать локальный Bluetooth-адаптер с удаленным устройством.

Пользователь должен явным образом разрешить взаимодействие двух устройств — либо через экран настроек Bluetooth, либо через диалоговое окно в вашем приложении, когда вы пытаетесь создать сокет для связи между двумя неспаренными адаптерами.

Вы можете установить канал связи по протоколу RFCOMM для двунаправленного обмена данными, используя классы:

- BluetoothServerSocket — нужен для создания ожидающего сокета, чтобы установить связь между двумя устройствами, при этом одно из устройств выступает в роли сервера, ожидая и принимая запрос на соединение;
- BluetoothSocket — применяется для создания нового клиентского сокета, чтобы соединиться с ожидающим объектом BluetoothServerSocket; как только связь будет установлена, оба сокета, на стороне клиента и сервера, могут быть использованы для передачи и приема потоков данных.

В процессе создания приложения, использующего Bluetooth в качестве канала связи, вам необходимо реализовать и BluetoothServerSocket, чтобы иметь возможность ожидать соединение, и BluetoothSocket, чтобы создавать новые каналы и управлять связью.

При подключении сокет BluetoothServerSocket возвращает объект BluetoothSocket, который впоследствии используется серверным устройством для передачи и приема данных. Этот BluetoothSocket, работающий на стороне сервера, может быть использован точно так же, как и клиентский сокет. Понятия «сервер» и «клиент» актуальны только на момент установления связи. Далее они никак не влияют на способ передачи данных.

Установка сокета BluetoothServerSocket в режим ожидания

BluetoothServerSocket используется для ожидания запроса на подключение от объекта BluetoothSocket, расположенного на удаленном устройстве. Чтобы установить связь между двумя адаптерами, один из них должен вести себя, как сервер (ожидая и принимая запросы на подключение), а другой — как клиент (инициируя запрос на подключение к серверу).

При подключении управление связью на обоих концах ведется с помощью объектов BluetoothSocket.

Чтобы начать процесс ожидания входящего соединения, вызовите метод listenUsingRfcommWithServiceRecord из объекта BluetoothAdapter, передавая ему в качестве параметров строку name (для идентификации вашего сервера) и универсальный уникальный идентификатор устройства UUID. В результате этих действий вы получите объект BluetoothServerSocket. Обратите внимание, что клиентский сокет, который будет подключаться к данному адаптеру, должен знать его UUID.

Начать отслеживание входящих подключений поможет метод accept, вызванный из серверного сокета (при необходимости можно также указать длительность этого процесса). Теперь BluetoothServerSocket будет заблокирован, пока к нему не попытается подключиться удаленное устройство, передающее соответствующий UUID. Если подключение инициировано

с устройства, которое еще не было связано с локальным адаптером, пользователю предложат разрешить связывание устройств, прежде чем метод accept закончит работу. Это предложение показано на рис. 13.3 и реализовано в виде уведомления.

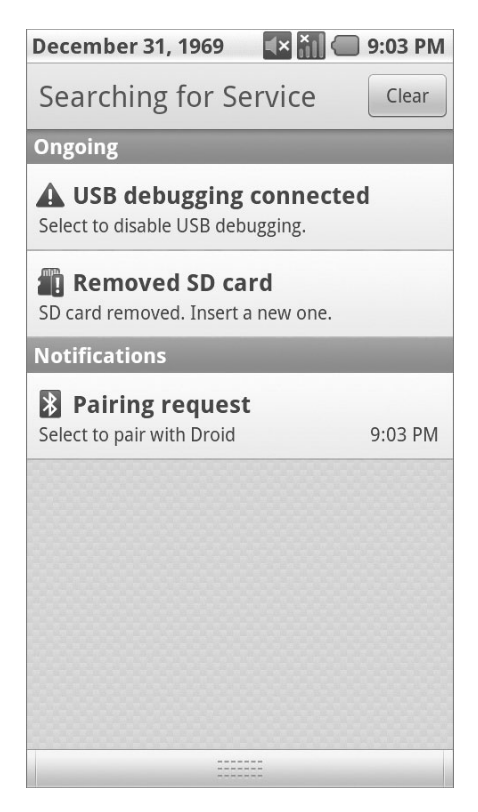


Рис. 13.3.

Если запрос на входящее подключение завершился успешно, метод accept вернет объект BluetoothSocket, подключенный к клиентскому устройству. Вы можете использовать его для передачи данных — это продемонстрировано в данном разделе.

ВНИМАНИЕ

Метод accept блокирует работу приложения, поэтому ожидание входящих подключений рекомендуется осуществлять в фоновом режиме, вместо того чтобы «замораживать» пользовательский интерфейс, пока соединение не будет установлено.

Важно отметить, что для установления связи ваш адаптер должен быть доступным для обнаружения удаленными Bluetooth-устройствами. В листинге 13.8 показан каркас, в котором ACTION_REQUEST_DISCOVERABLE

используется для того, чтобы узнать, доступен ли адаптер для сканирования, и только затем инициируется ожидание запросов на подключение, длящееся на протяжении времени, значение которого передается в параметре resultCode.

Листинг 13.8. Ожидание запросов на подключение от BluetoothSocket

```
startActivityForResult(new
  Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE),
         DISCOVERY_REQUEST);

@Override
protected void onActivityResult(int requestCode,
  int resultCode, Intent data) {
  if (requestCode == DISCOVERY_REQUEST) {
    boolean isDiscoverable = resultCode > 0;
    int discoverableDuration = resultCode;
    if (isDiscoverable) {
      UUID uuid = UUID.fromString("a60f35f0-b93a-11de-8a39-
08002009c666");
      String name = "bluetoothserver";

      final BluetoothServerSocket btserver =
        bluetooth.listenUsingRfcommWithServiceRecord(name, uuid);

      Thread acceptThread = new Thread(new Runnable() {
        public void run() {
          try {
            // Блокируется, пока соединение не будет установлено.
            BluetoothSocket serverSocket = btserver.accept();
           // TODO Передать данные с помощью серверного сокета
          } catch (IOException e) {
            Log.d("BLUETOOTH", e.getMessage());
          }
        }
      });
      acceptThread.start();
    }
  }
}
```

Выбор удаленных Bluetooth-устройств для подключения

Класс BluetoothSocket используется клиентским устройством для создания канала связи между вашим приложением и ожидающим серверным сокетом.

Создание клиентского сокета происходит с помощью вызова метода createRfcommSocketToServiceRecord из объекта BluetoothDevice. Этот объект представляет собой удаленное устройство, с которым необходимо связаться. На самом удаленном устройстве должен находиться объект BluetoothServerSocket, ожидающий запросов на подключение (описано в предыдущем разделе).

Есть несколько способов получить ссылку на удаленное устройство. Существуют также некоторые ограничения в отношении устройств, с которыми вы можете устанавливать связь.

Условия для установления соединения через Bluetooth. Чтобы объект BluetoothSocket мог установить соединение с удаленным Bluetooth-устройством, должны выполняться следующие условия:

- удаленное устройство должно быть доступным для обнаружения;
- удаленное устройство должно принимать подключения с помощью объекта BluetoothServerSocket;
- локальное и удаленное устройства должны быть спаренными (или связанными), иначе при установлении соединения пользователю будет предложено их связать.

Поиск Bluetooth-устройства для установления связи. Каждый экземпляр класса BluetoothDevice — удаленное устройство. Эти объекты используются для получения свойств удаленного адаптера и для соединения через BluetoothSocket. Существует несколько способов получить объект BluetoothDevice внутри своей программы.

В любом случае необходимо убедиться в том, что устройство, к которому вы хотите подключиться, доступно для обнаружения, и (при необходимости) узнать, привязано ли оно к вашему адаптеру. Если вы не можете обнаружить удаленное устройство, то предложите пользователю сделать его доступным для сканирования.

Ранее в этом разделе вы уже познакомились с одним способом обнаружения Bluetooth-устройств, который заключался в использовании метода startDiscovery и в отслеживании срабатывания действия ACTION_FOUND. Вы знаете, что каждое полученное Намерение содержит дополнительный параметр BluetoothDevice.EXTRA_DEVICE, в котором находится объект BluetoothDevice, представляющий обнаруженное устройство.

Можно использовать и метод getRemoteDevice из объекта BluetoothAdapter, указывая аппаратный адрес удаленного адаптера, к которому нужно подключиться.

```
BluetoothDevice device = bluetooth.getRemoteDevice("01:23:77:35:2F:AA");
```

Чтобы получить набор связанных на данный момент устройств, вызовите метод getBondedDevices из объекта BluetoothAdapter. Можно посылать запросы внутри возвращенного набора, чтобы проверить, связано ли целевое устройство с локальным адаптером.

```
Set<BluetoothDevice> bondedDevices = bluetooth.getBondedDevices();
if (bondedDevices.contains(remoteDevice))
// TODO Целевое устройство связано с локальным адаптером.
```

В листинге 13.9 представлен пример того, как можно проверить доступность целевого устройства для обнаружения и связанности.

Листинг 13.9. Проверка удаленного устройства на возможность обнаружения и связанность

```
final BluetoothDevice device =
                        bluetooth.getRemoteDevice("01:23:77:35:2F:AA");
final Set<BluetoothDevice> bondedDevices = bluetooth.getBondedDevices();

BroadcastReceiver discoveryResult = new BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent) {
  BluetoothDevice remoteDevice =
  intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

  if ((remoteDevice.equals(device) &&
  (bondedDevices.contains(remoteDevice)) {
  // TODO Целевое устройство связано с локальным адаптером
  // и доступно для обнаружения
  }
};
registerReceiver(discoveryResult,
                 new IntentFilter(BluetoothDevice.ACTION_FOUND));

if (!bluetooth.isDiscovering())
  bluetooth.startDiscovery();
```

Создание соединения с помощью клиентского сокета BluetoothSocket

Чтобы наладить канал связи с удаленным устройством, создайте сокет BluetoothSocket с помощью объекта BluetoothDevice, который это устройство представляет.

Для соединения вызовите метод createRfcommSocketToServiceRecord в контексте Bluetooth-устройства, к которому хотите подключиться, передав идентификатор UUID серверного сокета, ожидающего запрос на подключение.

Если вы попытаетесь подключиться к устройству, которое еще не было связано с вашим локальным адаптером, пользователю будет предложено осуществить связывание, прежде чем завершится вызов метода connect. Этот процесс показан на рис. 13.4.

Оба пользователя, на локальном и удаленном устройствах, должны разрешить связывание, чтобы соединение было установлено.

Объект BluetoothSocket, который при этом возвращается, может использоваться для подключения с помощью метода connect, как показано в листинге 13.10.

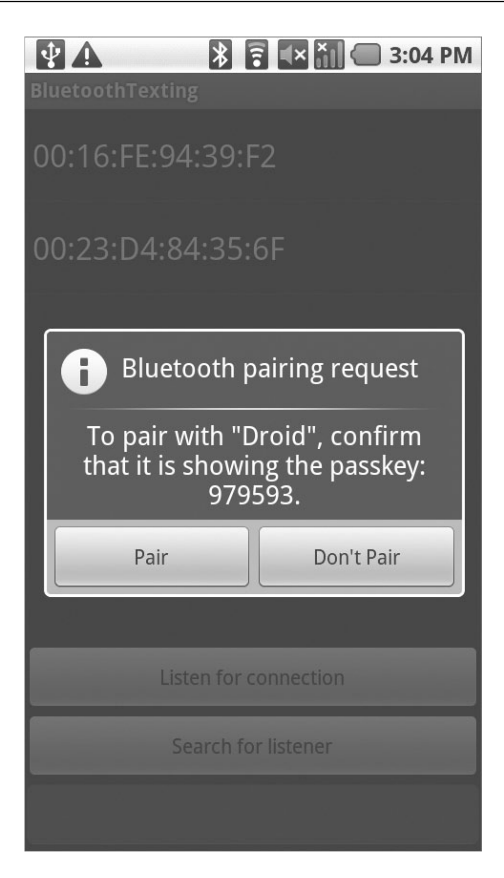


Рис. 13.4.

ВНИМАНИЕ

Метод connect блокирует работу приложения, поэтому запросы на подключение рекомендуется делать в фоновом режиме, вместо того чтобы «замораживать» пользовательский интерфейс, пока соединение не установлено.

Листинг 13.10. Соединение с удаленным серверным Bluetooth-устройством

```
try{
  BluetoothDevice device = bluetooth.getRemoteDevice("00:23:76:35:2F:AA");
  BluetoothSocket clientSocket =
    device.createRfcommSocketToServiceRecord(uuid);
  clientSocket.connect();
  // TODO Передача данных с помощью BluetoothSocket
} catch (IOException e) {
  Log.d("BLUETOOTH", e.getMessage());
}
```

Передача данных с помощью BluetoothSocket

Установив между собой соединение, оба устройства, клиентское и серверное, получат в распоряжение объекты BluetoothSocket. С этого момента все различия между ними нивелируются: с помощью объектов BluetoothSocket каждое устройство может отправлять и принимать данные.

Передача данных с помощью Bluetooth-сокетов управляется стандартными для Java объектами InputStream и OutputStream, которые вы можете получить из BluetoothSocket, используя методы getInputStream и getOutputStream соответственно.

В листинге 13.11 показаны два простых каркасных метода: первый используется для отправки строки на удаленное устройство с помощью OutputStream, второй — для ожидания входящих строк (с помощью InputStream). Такой подход можно применять для передачи любых потоковых данных.

Листинг 13.11. Отправка и получение строк с использованием BluetoothSocket

```
private void sendMessage(String message){
  OutputStream outStream;
  try {
    outStream = socket.getOutputStream();

    // Добавьте контрольный символ для остановки.
    byte[] byteArray = (message + " ").getBytes();
    byteArray[byteArray.length - 1] = 0;

    outStream.write(byteArray);
  } catch (IOException e) { }
}

private String listenForMessage()
  String result = "";
  int bufferSize = 1024;
  byte[] buffer = new byte[bufferSize];

  try {
    InputStream instream = socket.getInputStream();
    int bytesRead = -1;

    while (true) {
      bytesRead = instream.read(buffer);
      if (bytesRead != -1) {
        while ((bytesRead == bufferSize) && (buffer[bufferSize-1] != 0)){
          message = message + new String(buffer, 0, bytesRead);
          bytesRead = instream.read(buffer);
        }
        message = message + new String(buffer, 0, bytesRead - 1);
        return result;
      }
    }
  } catch (IOException e) {}

  return result;
}
```

Передача данных через Bluetooth

В следующем примере[1] с помощью API для работы с Bluetooth, предоставляемых платформой Android, создается простая одноранговая система обмена сообщениями между двумя спаренными Bluetooth-устройствами.

К сожалению, эмулятор Android на сегодняшний день не может применяться для тестирования возможностей Bluetooth. Чтобы проверить работу этого приложения, необходимы два физических устройства.

1. Начните с создания нового проекта BluetoothTexting, содержащего одноименную Активность. Добавьте в манифест два полномочия: BLUETOOTH и BLUETOOTH_ADMIN.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.paad.chapter13_bluetoothtexting"
  android:versionCode="1"
  android:versionName="1.0">
  <application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity
      android:name=".BluetoothTexting"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-sdk android:minSdkVersion="5" />
  <uses-permission android:name="android.permission.BLUETOOTH"/>
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
</manifest>
```

2. Отредактируйте ресурс с разметкой `main.xml`. Он должен содержать элемент ListView, отображающий список обнаруженных Bluetooth-устройств. Под ним должны находиться две кнопки: одна для запуска ожидающего BluetoothServerSocket, другая для подключения к ожидающему серверу.

   Добавьте элементы управления TextView и EditText, чтобы читать и передавать сообщения через установленное соединение.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
```

[1] Все фрагменты кода в этом примере — часть проекта Bluetooth Texting из главы 13, их можно загрузить с сайта Wrox.com.

```
    <EditText
      android:id="@+id/text_message"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:enabled="false"
    />
    <Button
      android:id="@+id/button_search"
      android:text="Search for listener" android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_above="@id/text_message"
    />
    <Button
      android:id="@+id/button_listen"
      android:text="Listen for connection"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_above="@id/button_search"
    />
    <ListView
      android:id="@+id/list_discovered"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_above="@id/button_listen"
      android:layout_alignParentTop="true"
    />
    <TextView
      android:id="@+id/text_messages"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_above="@id/button_listen"
      android:layout_alignParentTop="true"
      android:visibility="gone"
    />
  </RelativeLayout>
```

3. Переопределите метод onCreate для Активности BluetoothTexting. Сделайте вызовы нескольких методов-заглушек, которые понадобятся для доступа к Bluetooth-устройству, и настройки элементов пользовательского интерфейса.

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.UUID;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.AsyncTask;
```

```
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnKeyListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class BluetoothTexting extends Activity {

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  // Получите объект BluetoothAdapter
  configureBluetooth();

  // Настройте элемент ListView, который будет содержать обнаруженные
  // устройства
  setupListView();

  // Настройке кнопку для поиска устройств
  setupSearchButton();

  // Настройте кнопку для начала ожидания
  setupListenButton();
  }

  private void configureBluetooth() {}
  private void setupListenButton() {}
  private void setupListView() {}
  private void setupSearchButton() {}
}
```

4. Наполните кодом заглушку configureBluetooth, чтобы получить доступ к локальному Bluetooth-адаптеру и сохранить его в виде поля класса (его предварительно нужно создать). Оно будет содержать либо серверный, либо клиентский (если соединение было установлено) сокет. Вам также необходимо создать поле для UUID, чтобы идентифицировать приложение при подключении.

```
private BluetoothAdapter bluetooth;
private BluetoothSocket socket;
private UUID uuid = UUID.fromString("a60f35f0-b93a-11de-8a39-
08002009c666");

private void configureBluetooth() {
  bluetooth = BluetoothAdapter.getDefaultAdapter();
}
```

5. Создайте новый метод switchUI, который будет вызываться при установлении соединения, чтобы сделать возможным использование Представлений для чтения и отправки сообщений.

```
private void switchUI() {
  final TextView messageText = (TextView)findViewById(R.id.text_
messages);
  final EditText textEntry = (EditText)findViewById(R.id.text_message);

  messageText.setVisibility(View.VISIBLE);
  list.setVisibility(View.GONE);
  textEntry.setEnabled(true);
}
```

6. Заполните заглушку setupListenButton, создав серверный сокет для ожидания соединения. При вызове этого метода пользователю предложат сделать устройство доступным для обнаружения. После закрытия диалогового окна откройте BluetoothServerSocket для ожидания запросов на подключение (на протяжении определенного времени). Как только соединение установится, вызовите метод switchUI, который вы создали в пункте 5.

```
private static int DISCOVERY_REQUEST = 1;

private void setupListenButton() {
Button listenButton = (Button)findViewById(R.id.button_listen);
listenButton.setOnClickListener(new OnClickListener() {
public void onClick(View view) {
intent disc;
disc = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
startActivityForResult(disc, DISCOVERY_REQUEST);
}
});
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
  if (requestCode == DISCOVERY_REQUEST) {
    boolean isDiscoverable = resultCode > 0;
    if (isDiscoverable) {
      String name = "bluetoothserver";
      try {
        final BluetoothServerSocket btserver =
        bluetooth.listenUsingRfcommWithServiceRecord(name, uuid);

        AsyncTask<Integer, Void, BluetoothSocket> acceptThread =
          new AsyncTask<Integer, Void, BluetoothSocket>() {

          @Override
          protected BluetoothSocket doInBackground(Integer ... params) {
            try {
              socket = btserver.accept(params[0]*1000);
              return socket;
```

```
          } catch (IOException e) {
            Log.d("BLUETOOTH", e.getMessage());
          }
          return null;
        }
        @Override
        protected void onPostExecute(BluetoothSocket result) {
          if (result != null)
            switchUI();
        }
      };
      acceptThread.execute(resultCode);
    } catch (IOException e) {
      Log.d("BLUETOOTH", e.getMessage());
    }
  }
 }
}
```

7. Создайте клиентский код для подключения. С его помощью клиентское устройство сможет искать ожидающий сервер, сканируя и выводя на экран все доступные устройства.

   7.1. Начните с создания поля для хранения списка обнаруженных Bluetooth-устройств.

```
private ArrayList<BluetoothDevice> foundDevices;
```

   7.2. Затем наполните кодом заглушку setupListView. Создайте новый ArrayAdapter, связывающий элемент ListView со списком обнаруженных устройств.

```
private ArrayAdapter<BluetoothDevice> aa;
private ListView list;

private void setupListView() {
  aa = new ArrayAdapter<BluetoothDevice>(this,
                android.R.layout.simple_list_item_1,
                 foundDevices);
  list = (ListView)findViewById(R.id.list_discovered);
  list.setAdapter(aa);
}
```

   7.3. Создайте новый Приемник широковещательных намерений, который отслеживает действие BluetoothDevice.EXTRA_DEVICE и добавляет каждое обнаруженное устройство в список, созданный на шаге 7.1, и оповещает ArrayAdapter, созданный в предыдущем шаге.

```
BroadcastReceiver discoveryResult = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    BluetoothDevice remoteDevice;
    remoteDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
    if (bluetooth.getBondedDevices().contains(remoteDevice)) {
```

```
        foundDevices.add(remoteDevice);
        aa.notifyDataSetChanged();
      }
    }
  };
```

7.4. Закончите заглушку setupSearchButton, зарегистрировав объект BroadcastReceiver из предыдущего пункта и инициировав процесс обнаружения.

```
private void setupSearchButton() {
  Button searchButton = (Button)findViewById(R.id.button_search);

  searchButton.setOnClickListener(new OnClickListener() {
    public void onClick(View view) {
      registerReceiver(discoveryResult,
                       new IntentFilter(BluetoothDevice.ACTION_FOUND));

      if (!bluetooth.isDiscovering()) {
        foundDevices.clear();
        bluetooth.startDiscovery();
      }
    }
  });
}
```

8. Чтобы завершить написание кода, контролирующего процесс подключения, расширьте метод setupListView из пункта 7.2. Добавьте в него обработчик onItemClickListener, который в асинхронном режиме будет пытаться инициировать подключение клиентского адаптера к выбранному удаленному Bluetooth-устройству. В случае успеха сохраните ссылку на создающийся при этом сокет и сделайте вызов метода switchUI, созданного в пункте 5.

```
private void setupListView() {
  aa = new ArrayAdapter<BluetoothDevice>(this,
  android.R.layout.simple_list_item_1,
  foundDevices);
  list = (ListView)findViewById(R.id.list_discovered);
  list.setAdapter(aa);

  list.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> arg0, View view,
                            int index, long arg3) {
      AsyncTask<Integer, Void, Void> connectTask =
        new AsyncTask<Integer, Void, Void>() {
          @Override
          protected Void doInBackground(Integer ... params) {
            try {
              BluetoothDevice device = foundDevices.get(params[0]);
              socket = device.createRfcommSocketToServiceRecord(uuid);
              socket.connect();
            } catch (IOException e) {
              Log.d("BLUETOOTH_CLIENT", e.getMessage());
            }
```

```
          return null;
        }

        @Override
        protected void onPostExecute(Void result) {
          switchViews();
        }
      };
    connectTask.execute(index);
  }
});
}
```

9. Запустив это приложение на двух устройствах, можете нажать кнопку Listen for connection на одном и Search for listener на другом. В элементе ListView должны отобразиться все связанные адаптеры поблизости, как показано на рис. 13.5.

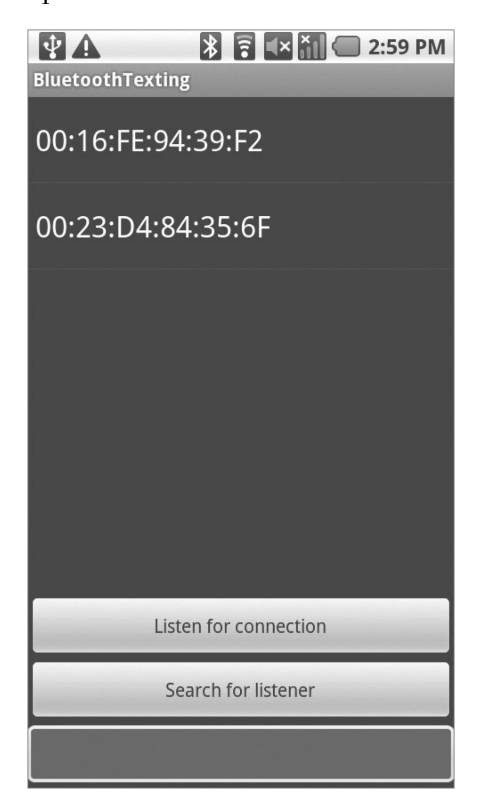


Рис. 13.5.

Если вы выберете устройство из списка, между ним и вашим локальным адаптером будет установлено соединение. В следующих пунктах этот канал связи понадобится для обмена простыми текстовыми сообщениями.

10. Начните с расширения метода switchUI. Добавьте новый обработчик нажатий к элементу EditText, чтобы следить за событиями манипулятора D-pad (джойстика). Как только такое событие произойдет, получите текст, содержащийся в элементе EditText, и отправьте его через BluetoothSocket.

```
private void switchUI() {
  final TextView messageText = (TextView)findViewById(R.id.text_
messages);
  final EditText textEntry = (EditText)findViewById(R.id.text_message);

  messageText.setVisibility(View.VISIBLE);
  list.setVisibility(View.GONE);
  textEntry.setEnabled(true);

  textEntry.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
      if ((keyEvent.getAction() == KeyEvent.ACTION_DOWN) &&
          (keyCode == KeyEvent.KEYCODE_DPAD_CENTER)) {
        sendMessage(socket, textEntry.getText().toString());
        textEntry.setText("");
        return true;
      }
      return false;
    }
  });
}

private void sendMessage(BluetoothSocket socket, String msg) {
  OutputStream outStream;
  try {
    outStream = socket.getOutputStream();
    byte[] byteString = (msg + " ").getBytes();
    stringAsBytes[byteString.length - 1] = 0;
    outStream.write(byteString);
  } catch (IOException e) {
    Log.d("BLUETOOTH_COMMS", e.getMessage());
  }
}
```

11. Чтобы принять отосланное сообщение, потребуется создать объект BluetoothSocketListener, работающий в асинхронном режиме и следящий за тем, принимает ли данные BluetoothSocket.

    11.1. Начните с нового класса MessagePoster, реализующего интерфейс Runnable. Его конструктор должен принимать два параметра: элемент TextView и строку с сообщением. Полученное сообщение нужно вставить в TextView. Этот класс будет использоваться для добавления входящих сообщений в пользовательский интерфейс из фонового потока.

```
private class MessagePoster implements Runnable {
  private TextView textView;
```

```
  private String message;

  public MessagePoster(TextView textView, String message) {
    this.textView = textView;
    this.message = message;
  }

  public void run() {
    textView.setText(message);
  }
}
```

11.2. Теперь создайте BluetoothSocketListener, реализующий интерфейс Runnable. Его конструктор должен принимать объект BluetoothSocket (который и будет отслеживаться), элемент TextView для отображения входящих сообщений и объект Handler для синхронизации при обновлении пользовательского интерфейса.

Получив новое сообщение, отобразите его в элементе TextView, используя объект MessagePoster из предыдущего пункта.

```
private class BluetoothSocketListener implements Runnable {

  private BluetoothSocket socket;
  private TextView textView;
  private Handler handler;

  public BluetoothSocketListener(BluetoothSocket socket,
                                 Handler handler, TextView textView) {
    this.socket = socket;
    this.textView = textView;
    this.handler = handler;
  }

  public void run() {
    int bufferSize = 1024;
    byte[] buffer = new byte[bufferSize];
    try {
      InputStream instream = socket.getInputStream();
      int bytesRead = -1;
      String message = "";
      while (true) {
        message = "";
        bytesRead = instream.read(buffer);
        if (bytesRead != -1) {
          while ((bytesRead==bufferSize)&&(buffer[bufferSize-1] != 0)) {
            message = message + new String(buffer, 0, bytesRead);
            bytesRead = instream.read(buffer);
          }
          message = message + new String(buffer, 0, bytesRead - 1);

          handler.post(new MessagePoster(textView, message));

          socket.getInputStream();
```

```
        }
      }
    } catch (IOException e) {
      Log.d("BLUETOOTH_COMMS", e.getMessage());
    }
  }
}
```

11.3. В завершение сделайте еще одно дополнение к методу swichUI, создав и запустив новый объект BluetoothSocketListener из предыдущего пункта.

```
private Handler handler = new Handler();

private void switchUI() {
  final TextView messageText = (TextView)findViewById(R.id.text_
messages);
  final EditText textEntry = (EditText)findViewById(R.id.text_message);

  messageText.setVisibility(View.VISIBLE);
  list.setVisibility(View.GONE);
  textEntry.setEnabled(true);

  textEntry.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
      if ((keyEvent.getAction() == KeyEvent.ACTION_DOWN) &&
          (keyCode == KeyEvent.KEYCODE_DPAD_CENTER)) {
        sendMessage(socket, textEntry.getText().toString());
        textEntry.setText("");
        return true;
      }
      return false;
    }
  });


  BluetoothSocketListener bsl = new BluetoothSocketListener(socket,
                                      handler, messageText);
  Thread messageListener = new Thread(bsr);
  messageListener.start();
}
```

Теперь, запустив приложение, вы сможете настроить одно устройство на ожидание подключения, а другое — на инициирование соединения. Как только связь установится, можно обмениваться текстовыми сообщениями.

Имейте в виду, что данный пример максимально упрощен, чтобы основной акцент сделать на описание принципов работы с Bluetooth. Для улучшения реализации этого приложения можно перенести весь код, отвечающий за соединение, в отдельный Сервис, отменяя регистрацию Приемника широковещательных намерений после взаимодействия устройств и установления между ними связи.

Управление сетевыми соединениями

Невероятный рост интернет-услуг и распространенность мобильных устройств привели к тому, что беспроводной доступ к Сети становится все более популярным на мобильных телефонах.

Поскольку скорость, надежность и стоимость подключения к Интернету зависят от сетевой технологии, с помощью которой устанавливается соединение (Wi-Fi, GPRS, 3G), отслеживание этих соединений и управление ими может помочь улучшить надежность и отзывчивость ваших приложений.

Android предоставляет Намерения, которые несут информацию об изменениях в подключении к Сети, и предлагает API для контроля за сетевыми настройками и соединениями.

Важно, что пользователи могут задавать собственные сетевые настройки, особенно при передаче данных в фоновом режиме.

Управляет сетью в Android главным образом Сервис ConnectivityManager, он позволяет отслеживать состояние подключения к Сети, устанавливать предпочтительное сетевое подключение и контролировать отказоустойчивость соединения.

Позже вы узнаете, как использовать класс WifiManager для отслеживания и изменения подключений по Wi-Fi. WifiManager позволяет создавать новые конфигурации для Wi-Fi, отслеживать и изменять уже существующие сетевые настройки, управлять активными подключениями, а также проводить сканирование точек доступа.

Знакомство с Сервисом ConnectivityManager

ConnectivityManager — Сервис для управления сетевыми подключениями, используемый для отслеживания состояния подключений к Сети, настройки отказоустойчивости и управления сетевыми адаптерами.

Для того чтобы получить доступ к ConnectivityManager, примените метод getSystemService, передав ему константу Context.CONNECTIVITY_SERVICE в качестве имени Сервиса, как показано в листинге 13.12.

Листинг 13.12. Получение доступа к ConnectivityManager

```
String service = Context.CONNECTIVITY_SERVICE;
ConnectivityManager connectivity =
  (ConnectivityManager)getSystemService(service);
```

Чтобы использовать ConnectivityManager, ваше приложение должно иметь полномочия на чтение и запись состояния Сети. Добавьте их в манифест, как показано ниже:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
```

Чтение пользовательских настроек, отвечающих за передачу данных в фоновом режиме

Один из наиболее важных наборов информации, доступных через ConnectivityManager, — пользовательские настройки для передачи данных в фоновом режиме.

Пользователи могут включать или выключать фоновую передачу данных, перейдя в меню Settings ▸ Accounts & sync settings ▸ Background data setting, как показано на рис. 13.6.

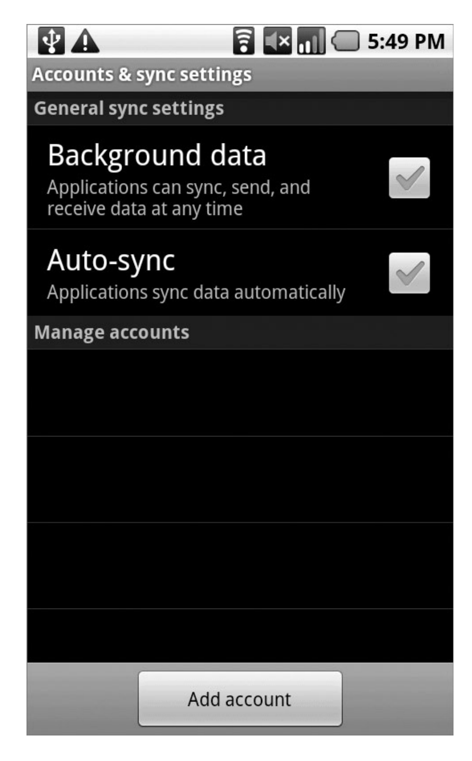


Рис. 13.6.

Данное значение контролируется на уровне приложения, это значит, что вы как разработчик отвечаете за его считывание и соблюдение в контексте пользовательских настроек, разрешая передачу данных в фоновом режиме.

Чтобы получить эти настройки, вызовите метод getBackgroundDataSetting из объекта ConnectivityManager:

```
boolean backgroundEnabled = connectivity.getBackgroundDataSetting();
```

Если эта возможность выключена, ваше приложение должно передавать данные только в том случае, когда оно в активном состоянии и на переднем плане. Выключая эту функцию, пользователь явно просит, чтобы ваше приложение, находясь в фоновом режиме, не передавало информацию по Сети.

Если для функционирования вашего приложения необходима возможность передачи данных в фоновом режиме, лучше уведомить пользователя о таком требовании и предложить ему поменять свои настройки.

Когда пользователь изменит настройки, отвечающие за передачу данных в фоновом режиме, система пошлет Широковещательное намерение с действием ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED.

Чтобы следить за изменениями состояния этой настройки, создайте и зарегистрируйте объект BroadcastReceiver, который будет реагировать на соответствующее Намерение, как показано в листинге 13.13.

Листинг 13.13. Получение доступа к ConnectivityManager

```
registerReceiver(
  new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
    // Действия при изменении настроек, отвечающих за
    // передачу данных в фоновом режиме.
    },
  new IntentFilter(ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_
CHANGED));
```

ПРИМЕЧАНИЕ

Ваши приложения не обязаны учитывать пользовательские настройки для передачи данных в фоновом режиме. Однако тогда вы рискуете услышать громкую критику в свой адрес, а пользователь может получить огромный счет за сетевой трафик.

Отслеживание информации о сети

Сервис ConnectivityManager предоставляет высокоуровневый доступ к имеющимся сетевым подключениям. Используя методы getActiveNetworkInfo или getNetworkInfo, как показано в листинге 13.14, вы можете получить объект NetworkInfo, содержащий информацию о сетевом подключении, активном в данный момент, или о неактивной сети заданного типа.

Используйте NetworkInfo для получения сведений о состоянии подключения, о типе сети, а также подробного описания состояния сетевого соединения.

Листинг 13.14. Получение доступа к информации о сети

```
// Получите информацию об активном сетевом подключении.
NetworkInfo activeNetwork = connectivity.getActiveNetworkInfo();
int networkType = networkInfo.getType();
switch (networkType) {
  case (ConnectivityManager.TYPE_MOBILE) : break;
  case (ConnectivityManager.TYPE_WIFI) : break;
  default: break;
}

// Получите информацию о мобильной сети.
int network = ConnectivityManager.TYPE_MOBILE;
NetworkInfo mobileNetwork = connectivity.getNetworkInfo(network);


NetworkInfo.State state = mobileNetwork.getState();
NetworkInfo.DetailedState detailedState = mobileNetwork.
getDetailedState();
```

Поиск и изменение сетевых настроек, управление аппаратными адаптерами

ConnectivityManager может использоваться для управления сетевым аппаратным обеспечением и настройками отказоустойчивости.

Android будет пытаться подключиться к предпочтительной сети каждый раз, когда авторизованное приложение запросит доступ к интернет-соединению. Вы можете получить текущее соединение (или установить предпочтительное) с помощью методов getNetworkPreference и setNetworkPreference, как показано в следующем фрагменте кода:

```
int networkPreference = connectivity.getNetworkPreference();
connectivity.setNetworkPreference(NetworkPreference.PREFER_WIFI);
```

Если предпочтительное подключение недоступно или соединение с этой сетью было утрачено, Android автоматически попытается подключиться ко второстепенной сети.

Вы можете контролировать доступность сети определенного типа, используя метод setRadio. С помощью него есть возможность изменять состояние адаптера, привязанного к конкретной сети (Wi-Fi, мобильной сети и т. д.). Например, в следующем фрагменте кода адаптер для Wi-Fi выключен и вместо него включится адаптер для мобильной сети:

```
connectivity.setRadio(NetworkType.WIFI, false);
connectivity.setRadio(NetworkType.MOBILE, true);
```

Отслеживание состояния сетевого соединения

Одной из наиболее полезных функций сервиса ConnectivityManager — уведомление приложений об изменениях в сетевых подключениях.

Чтобы отслеживать состояние подключения к сети, создайте собственную реализацию класса BroadcastReceiver, которая должна принимать Намерения с действием ConnectivityManager.CONNECTIVITY_ACTION. Эти Намерения содержат несколько дополнительных параметров, предоставляющих расширенную информацию об изменениях соединения. Вы можете получить доступ к каждому из них, используя статические константы из класса ConnectivityManager.

- EXTRA_IS_FAILOVER. Возвращает true, если текущее соединение выбрано в результате возникновения проблем с предпочтительной сетью.
- EXTRA_NO_CONNECTIVITY. Возвращает true, если устройство не подключено ни к какой сети.
- EXTRA_REASON. Если соответствующее Намерение свидетельствует о разрыве связи, в этом строковом значении будет содержаться описание того, почему попытка подключения не удалась.
- EXTRA_NETWORK_INFO. Возвращает объект NetworkInfo, содержащий более подробные данные о сети, связанной с текущим событием.
- EXTRA_OTHER_NETWORK_INFO. После отключения от сети это значение вернет объект NetworkInfo, содержащий информацию для возможного резервного соединения.
- EXTRA_EXTRA_INFO. Несет дополнительные, специфичные для конкретного типа сети данные о соединении.

Управление подключением к сети Wi-Fi

Класс WifiManager представляет собой сервис для работы с Wi-Fi в Android. Он может быть использован для настройки соединений, установленных через Wi-Fi, управления текущим соединением этого типа, сканирования точек доступа и отслеживания изменений в подключении к Wi-Fi.

Как и в случае с ConnectivityManager, получить доступ к WifiManager можно с помощью метода getSystemService, в который должна быть передана константа Context.WIFI_SERVICE, как показано в листинге 13.15.

Листинг 13.15. Получение доступа к WifiManager

```
String service = Context.WIFI_SERVICE;
WifiManager wifi = (WifiManager)getSystemService(service);
```

Чтобы использовать WifiManager, ваше приложение должно иметь полномочия для доступа к сетям Wi-Fi и изменения их состояния. Для этого нужно добавить следующие строки в манифест:

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

Можно задействовать WifiManager для включения и выключения аппаратного адаптера Wi-Fi, используя метод setWifiEnabled, или получать текущее состояние Wi-Fi с помощью методов getWifiState или isWifiEnabled, как показано в листинге 13.16.

Листинг 13.16. Отслеживание и изменение состояние Wi-Fi

```
if (!wifi.isWifiEnabled())
  if (wifi.getWifiState() != WifiManager.WIFI_STATE_ENABLING)
    wifi.setWifiEnabled(true);
```

Следующие несколько разделов посвящены отслеживанию состояния текущего соединения по Wi-Fi и способу изменения уровня сигнала. Позже вы также научитесь сканировать точки доступа и подключаться к ним.

Этих возможностей вполне достаточно для большинства разработчиков, но WifiManager также предоставляет низкоуровневый доступ к параметрам сетей Wi-Fi. Вы можете получить полный контроль над всеми настройками Wi-Fi, что при необходимости позволит полностью заменить системное приложение для управления данным типом сети. ниже в этом разделе вы бегло ознакомитесь с программными интерфейса, которые используются для удаления и изменения сетевых конфигураций.

Отслеживание соединения по Wi-Fi

WifiManager транслирует Намерения каждый раз, когда состояние подключения к сети Wi-Fi изменяется. При этом используется действие, заданное с помощью одной из констант класса WifiManager.

- WIFI_STATE_CHANGED_ACTION. Сигнализирует, что аппаратный статус Wi-Fi изменен, оперируя такими состояниями, как включен, выключается, выключен и неизвестно. Намерение содержит два дополнительных параметра с ключами EXTRA_WIFI_STATE и EXTRA_PREVIOUS_STATE, которые предоставляют текущее и предыдущее состояние соответственно.
- SUPPLICANT_CONNECTION_CHANGE_ACTION. Это Намерение передается каждый раз, когда меняется состояние соединения с активной точкой доступа. Оно срабатывает, если установлено новое соединение или уже существующее разорвано. В первом случае дополнительный параметр EXTRA_NEW_STATE будет иметь значение true.
- NETWORK_STATE_CHANGED_ACTION. Срабатывает при изменении состояния подключения к сети Wi-Fi. Это Намерение содержит два дополнительных параметра: первый, EXTRA_NETWORK_INFO, включает объект NetworkInfo, описывающий текущее состояние соединения, второй, EXTRA_BSSID, передает идентификатор BSSID, принадлежащий точке доступа, к которой вы подключены.

- RSSI_CHANGED_ACTION. Вы можете отслеживать текущий уровень сигнала в сети Wi-Fi, к которой подключены, используя Намерение с действием RSSI_CHANGED_ACTION. Оно содержит дополнительный целочисленный параметр EXTRA_NEW_RSSI с показателем уровня сигнала. Чтобы воспользоваться этим значением, необходимо конвертировать его с учетом указанной шкалы, вызвав статический метод calculateSignalLevel из объекта WifiManager.

Получение информации об активном соединении

Чтобы получить информацию о статусе только что установленного активного сетевого соединения, используйте метод getConnectionInfo из объекта WifiManager. Возвращаемый этим методом объект WifiInfo содержит идентификаторы SSID, BSSID, MAC- и IP-адрес текущей точки доступа, а также скорость соединения и уровень сигнала.

В листинге 13.17 демонстрируется процесс получения информации об активном соединении, работающем через Wi-Fi.

Листинг 13.17. Запросы к активному сетевому соединению

```
WifiInfo info = wifi.getConnectionInfo();
if (info.getBSSID() != null) {
  int strength = WifiManager.calculateSignalLevel(info.getRssi(), 5);
  int speed = info.getLinkSpeed();
  String units = WifiInfo.LINK_SPEED_UNITS;
  String ssid = info.getSSID();

  String cSummary = String.format("Connected to %s at %s%s. Strength %s/5",
                                  ssid, speed, units, strength);
}
```

Обнаружение точек доступа

Вы также можете использовать WifiManager для сканирования на наличие точек доступа. Для этого предусмотрен метод startScan.

Намерение с действием SCAN_RESULTS_AVAILABLE_ACTION будет транслироваться в асинхронном режиме, объявляя о завершении сканирования и доступности результатов.

Вызовите метод getScanResults, чтобы получить эти результаты в виде списка объектов типа ScanResult. Все они будут содержать информацию об обнаруженных точках доступа (скорость соединения, уровень сигнала, идентификатор SSID и поддерживаемые методы аутентификации).

В листинге 13.18 показано, как инициировать обнаружение точек доступа, в ходе которого с помощью уведомлений типа Toast отображаются количество найденных точек и имя одной из них — той, у которой самый высокий уровень сигнала.

Листинг 13.18. Обнаружение точек доступа

```
// Зарегистрируйте объект BroadcastReceiver, отслеживающий результаты
сканирования.
registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    List<ScanResult> results = wifi.getScanResults();
    ScanResult bestSignal = null;
    for (ScanResult result : results) {
      if (bestSignal == null ||
          WifiManager.compareSignalLevel(bestSignal.level,result.
level)<0)
        bestSignal = result;
    }

    String toastText = String.format("%s networks found. %s is
                                      the strongest.",
                                      results.size(), bestSignal.SSID);

    Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_
LONG);
  }
}, new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));

// Запустите процесс сканирования.
wifi.startScan();
```

Управление настройками Wi-Fi

Вы можете использовать WifiManager для управления настройками установленного подключения и выбирать, с какой сетью устанавливать соединение. Подключившись к сети, можно узнать дополнительную информацию и ее настройки.

Получите список текущих сетевых конфигураций с помощью метода getConfiguredNetworks. Результат будет состоять из объектов типа WifiConfiguration, включающих идентификатор сети, SSID и другие подробности о каждой конфигурации.

Чтобы использовать конкретную конфигурацию сети, вызовите метод enableNetwork, передав ему сетевой идентификатор и значение true для параметра disableAllOthers, как показано в листинге 13.19.

Листинг 13.19. Активизация сетевого подключения

```
// Получите список доступных конфигураций
List<WifiConfiguration> configurations = wifi.getConfiguredNetworks();
// Получите идентификатор для первой сети из списка.
if (configurations.size() > 0) {
  int netID = configurations.get(0).networkId;
  // Подключитесь к этой сети.
  boolean disableAllOthers = true;
  wifi.enableNetwork(netID, disableAllOtherstrue);
}
```

Создание конфигураций для сети Wi-Fi

Чтобы подключиться к сети по Wi-Fi, необходимо создать и зарегистрировать для нее конфигурацию. Как правило, ваши пользователи делают это сами с помощью системных настроек для Wi-Fi. Но ничего не мешает добавить эту возможность в собственное приложение или даже полностью заменить стандартную Активность для управления конфигурацией Wi-Fi.

Сетевые конфигурации хранятся в виде объектов типа WifiConfiguration. Приведем список некоторых публичных полей, которые предоставляет этот класс:

- BSSID — идентификатор BSSID для точки доступа;
- SSID — идентификатор SSID конкретной сети;
- networkId — уникальный идентификатор, определяющий конфигурацию данной сети на текущем устройстве;
- priority — приоритет сетевых конфигураций для сортировки списка точек доступа, к которым можно подключиться;
- status — текущее состояние данного сетевого подключения; может иметь значение WifiConfiguration.Status.ENABLED, WifiConfiguration.Status.DISABLED или WifiConfiguration.Status.CURRENT.

Объект WifiConfiguration также содержит список поддерживаемых методов аутентификации и ключи, использованные ранее для проверки подлинности при подключении к этой точке доступа.

Метод addNetwork позволяет добавлять новую конфигурацию в текущий список. Аналогичным образом с помощью метода updateNetwork вы можете обновить сетевую конфигурацию, передав в качестве параметра объект WifiConfiguration, содержащий идентификатор сети и значения, которые хотите изменить.

Можно воспользоваться методом removeNetwork для удаления конфигурации, передав в качестве параметра идентификатор сети.

Чтобы сохранить любые изменения, внесенные в сетевые настройки, нужно вызвать метод saveConfiguration.

Резюме

В этой главе вы узнали, как отслеживать и контролировать некоторые низкоуровневые функции для обеспечения связи, доступные в устройствах под управлением Android.

В данной главе рассмотрены принципы работы с Bluetooth, механизмы обеспечения связи, отслеживания и изменения соединения с Интернетом

и другими сетями. С помощью класса WifiManager вы научились контролировать подключение устройства к сети по Wi-Fi, считывая ее состояние и конфигурацию.

В следующей главе вы узнаете, каким образом с помощью взаимодействия с объектом SensorManager можно предоставить своим приложениям доступ к реальному миру. Научитесь работать с аппаратными сенсорами, в частности с компасом и акселерометром, а также отслеживать и интерпретировать данные, полученные с их помощью.

Глава 14
ДАТЧИКИ

Содержание главы

- Использование объекта SensorManager.
- Доступные типы датчиков.
- Отслеживание состояния датчиков и интерпретация их показаний.
- Использование компаса, акселерометра и датчиков ориентации.
- Преобразование стандартной системы координат при вычислении положения в пространстве.
- Управление вибрацией устройства.

Современные мобильные телефоны — не просто устройства для связи, подключенные к Интернету. Смартфоны с микрофонами, камерами, акселерометрами, компасами, датчиками температуры и яркости способны «воспринимать» окружающий мир, расширяя ваше собственное восприятие.

В следующих главах вы изучите работу с камерой и микрофоном, а в этой главе познакомитесь с датчиками внешней среды на устройствах под управлением Android.

Датчики, следящие за физическими свойствами и состоянием окружающей среды, предоставляют инновационные способы для улучшения мобильных приложений. Наличие в современных телефонах электронных компасов, датчиков равновесия, яркости и близости открывает целый ряд новых возможностей для взаимодействия с устройством, таких как дополненная реальность и ввод данных, основанный на перемещениях в пространстве.

В этой главе вы также познакомитесь с датчиками в Android и объектом SensorManager, отслеживающим их состояние. Подробно рассмотрите возможности акселерометра и датчиков ориентации, научитесь использовать их для определения изменений положения и ускорения устройства. Это может быть особенно полезно при создании интерфейсов, основанных на движении, это позволит добавлять новые измерения к вашим геолокационным приложениям.

Вы также узнаете, как управлять вибрацией устройства, чтобы использовать обратный отклик в своих приложениях.

Использование датчиков и объекта sensormanager

Класс SensorManager разработан для работы с аппаратными датчиками, доступными в устройствах под управлением Android. Используйте метод getSystemService, чтобы получить ссылку на объект SensorManager, как показано в следующем фрагменте:

```
String service_name = Context.SENSOR_SERVICE;
SensorManager sensorManager = (SensorManager)getSystemService(service_
name);
```

Знакомство с датчиками

Как и в случае с геолокационными сервисами, Android скрывает от разработчика реализацию датчиков на каждом конкретном устройстве. Класс Sensor применяется для описания свойств аппаратного датчика, включая его тип, название, производителя, а также информацию о его точности и диапазоне.

Данный класс содержит набор констант, используемых для определения, какой именно тип датчика представляет объект Sensor. Эти константы имеют вид Sensor.TYPE_<TYPE>. В следующих разделах будут рассмотрены все поддерживаемые типы датчиков, после чего вы научитесь получать к ним доступ и использовать их в своих приложениях.

Датчики, поддерживаемые платформой Android

Ниже представлены типы датчиков, доступные на сегодняшний день; имейте в виду, что набор датчиков, действительно доступных для вашего приложения, определяется аппаратной начинкой устройства.

- Sensor.TYPE_ACCELEROMETER. Трехосевой акселерометр, возвращающий текущее ускорение по трем осям в м/с . Более подробно этот датчик будет рассмотрен в следующих разделах данной главы.
- Sensor.TYPE_GYROSCOPE. Гироскоп (гироскопический датчик), возвращающий текущее положение устройства в пространстве в градусах по трем осям.
- Sensor.TYPE_LIGHT. Датчик окружающей освещенности, возвращающий одиночное значение, которое описывает внешнюю освещенность в люксах. Этот тип датчиков обычно используется для динамического изменения яркости экрана.

- Sensor.TYPE_MAGNETIC_FIELD. Датчик магнитного поля, определяющий текущие показатели магнитного поля в микротеслах по трем осям.
- Sensor.TYPE_ORIENTATION. Датчик ориентации, возвращающий положение устройства в градусах по трем осям. Более детально будет рассмотрен позже в этой главе.
- Sensor.TYPE_PRESSURE. Датчик давления, возвращающий одиночное значение — текущее давление в килопаскалях, оказываемое на устройство.
- Sensor.TYPE_PROXIMITY. Датчик приближенности, который сигнализирует о расстоянии между устройством и целевым объектом (в метрах). Каким образом выбирается объект и какие расстояния поддерживаются, зависит от аппаратной реализации данного датчика. Типичное его применение — обнаружение расстояния между устройством и ухом пользователя для автоматического регулирования яркости экрана или выполнения голосовой команды.
- Sensor.TYPE_TEMPERATURE. Термометр, возвращающий температуру в градусах Цельсия. В зависимости от реализации это может быть температура в помещении, удаленного датчика, показатель нагрева батареи.

Поиск датчиков

Устройство на базе Android может включать в себя несколько реализаций одного и того же типа датчиков. Чтобы найти реализацию, использующуюся по умолчанию, вызовите метод getDefaultSensor из объекта SensorManager, передавая ему в качестве параметра тип датчика в виде одной из констант, описанных в предыдущем разделе.

Следующий фрагмент кода вернет объект, описывающий гироскоп по умолчанию. Если для данного типа не существует датчика по умолчанию, будет возвращено значение null.

```
Sensor defaultGyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_
GYROSCOPE);
```

Вы можете использовать метод getSensorList для получения списка всех доступных датчиков конкретного типа. В следующем фрагменте кода будут возвращены объекты Sensor, представляющие собой все доступные датчики давления:

```
List<Sensor> pressureSensors = sensorManager.getSensorList(Sensor.TYPE_
PRESSURE);
```

Чтобы получить все датчики, доступные на устройстве, используйте метод getSensorList, передавая ему константу Sensor.TYPE_ALL, как показано ниже:

```
List<Sensor> allSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

Данный подход позволит обнаружить все датчики и их типы, доступные на устройстве.

Использование датчиков

В листинге 14.1 показан стандартный способ отслеживания результатов, передаваемых аппаратными сенсорами. В следующих разделах более подробно будут рассмотрены реализации датчика ориентации в пространстве и акселерометра.

Реализуйте интерфейс SensorEventListener. Используйте onSensorChanged для отслеживания значений, возвращаемых датчиком, и onAccuracyChanged для реагирования на изменения в их точности.

Листинг 14.1. Каркас для реализации SensorEventListener

```
final SensorEventListener mySensorEventListener = new
SensorEventListener() {
  public void onSensorChanged(SensorEvent sensorEvent) {
    // TODO Отслеживать изменения в показаниях датчиков.
  }

  public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // TODO Реакция на изменения в точности значений, выдаваемых датчиками.
  }
};
```

Параметр SensorEvent, который передается в метод onSensorChanged, содержит четыре свойства, описывающих событие, связанное с датчиком.

- sensor. Объект Sensor, инициировавший событие.
- accuracy. Точность датчика во время поступления события (низкая, средняя, высокая или недостоверная, о чем пойдет речь в следующем маркированном списке).
- values. Массив чисел типа float, содержащий новые полученные значения. В следующем разделе будут рассмотрены значения, возвращаемые для каждого типа датчика.
- timestamp. Время возникновения события, связанного с датчиком (в наносекундах).

Вы также можете отслеживать точность каждого отдельного датчика, используя метод onAccuracyChanged. Оба этих обработчика используют

значение accuracy для представления степени точности датчика с помощью одной из констант.

- SensorManager.SENSOR_STATUS_ACCURACY_LOW. Говорит о том, что данные, предоставляемые датчиком, имеют низкую точность и нуждаются в калибровке.
- SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM. Говорит о средней степени точности датчика и том, что калибровка может улучшить результат.
- SensorManager.SENSOR_STATUS_ACCURACY_HIGH. Показатели датчика точны настолько, насколько это возможно.
- SensorManager.SENSOR_STATUS_UNRELIABLE. Данные, предоставляемые датчиком, недостоверны. Это значит, что датчик необходимо откалибровать, иначе невозможно считывать результаты.

Чтобы получать события, генерируемые датчиками, зарегистрируйте свою реализацию интерфейса SensorEventListener с помощью SensorManager. Укажите объект Sensor, за которым вы хотите наблюдать, и частоту, с которой вам необходимо получать обновления. В следующем примере показан процесс регистрации SensorEventListener для датчика приближенности по умолчанию с указанием стандартной частоты обновления:

```
Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
sensorManager.registerListener(mySensorEventListener,
                               sensor,
                               SensorManager.SENSOR_DELAY_NORMAL);
```

Класс SensorManager содержит следующие константы для выбора подходящей частоты обновлений (в порядке убывания):

- SensorManager.SENSOR_DELAY_FASTEST — самая высокая возможная частота обновления показаний датчиков;
- SensorManager.SENSOR_DELAY_GAME — частота, используемая для управления играми;
- SensorManager.SENSOR_DELAY_NORMAL — частота обновлений по умолчанию;
- SensorManager.SENSOR_DELAY_UI — частота для обновления пользовательского интерфейса.

Выбранная вами частота необязательно будет соблюдаться; SensorManager может возвращать результаты быстрее или медленней, чем вы указали (хотя, как правило, это происходит быстрее). Чтобы минимизировать расход ресурсов при использовании датчиков в приложении, необходимо пытаться подбирать наиболее низкую частоту.

Важно также не забыть отменить регистрацию SensorEventListener после того, как ваше приложение больше не будет нуждаться в получении информации от датчиков.

```
sensorManager.unregisterListener(mySensorEventListener);
```

Хорошее решение — регистрировать и отменять регистрацию SensorEventListener в методах onResume и onPause Активности. Благодаря этому датчики будут использоваться только тогда, когда Активность видна на экране.

Интерпретация данных, полученных с помощью датчиков

Длина и содержание значений, полученных с помощью метода onSensorChanged, варьируются в зависимости от датчика, состояние которого вы отслеживаете.

Подробная информация приводится в табл. 14.1. Использование акселерометра, датчиков ориентации и магнитного поля более подробно будет рассмотрено в следующих разделах.

ПРИМЕЧАНИЕ

Документацию, в которой описываются значения, возвращаемые каждым типом датчиков, а также некоторые дополнительные комментарии вы можете найти по адресу http://developer.android.com/reference/android/hardware/Sensor.html.

Таблица 14.1. Значения, возвращаемые датчиками

Датчик	Количество значений	Содержание значений	Комментарий
TYPE_ ACCELEROMETER	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Ускорение в м/с^2 по трем осям. SensorManager содержит набор констант вида SensorManager. GRAVITY_*
TYPE_GYROSCOPE	3	value[0] : Azimuth value[1] : Pitch value[2] : Roll	Положение устройства в пространстве в градусах по трем осям
TYPE_ LIGHT	1	value[0] : Illumination	Измеряется в люксах. SensorManager содержит набор констант для представления разных стандартных степеней освещенности, имеющих вид SensorManager.LIGHT_*
TYPE_MAGNETIC_ FIELD	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Внешнее магнитное поле, измеряющееся в микротеслах (μT)

Датчик	Количество значений	Содержание значений	Комментарий
YPE_ORIENTATION	3	value[0] : Azimuth value[1] : Roll value[2] : Pitch	Положение устройства в пространстве в градусах по трем осям
TYPE_PRESSURE	1	value[0] : Pressure	Измеряется в килопаскалях (кПа)
TYPE_PROXIMITY	1	value[0] : Distance	Измеряется в метрах
TYPE_ TEMPERATURE	1	value[0] : Temperature	Измеряется в градусах Цельсия

Использование компаса, акселерометра и датчика ориентации

Использование внутри программ информации о движении и положении в пространстве стало возможным благодаря наличию во многих современных устройствах акселерометра и датчика ориентации.

В последние годы эти датчики все более распространены, они поставляются с игровыми приставками, такими как Nintendo Wii, и мобильными телефонами (Apple iPhone, Palm Pre и множество устройств на базе Android).

Акселерометры и компасы предоставляют данные о направлении, положении и движении устройства. В последнее время отмечается тенденция использовать эти функции для механизма ввода, альтернативного традиционным сенсорным экранам, трекболам и клавиатурам.

Наличие акселерометра и компаса определяется аппаратным обеспечением, на котором выполняется приложение. Если датчики доступны, то работа с ними ведется с помощью объекта SensorManager.

Это позволит:

- определять текущее направление устройства;
- отслеживать изменения в направлении;
- узнавать, в какую сторону смотрит пользователь;
- следить за изменениями в скорости движения в любом направлении: вертикальном, боковом или продольном.

Это открывает для ваших приложений новые возможности. Отслеживая положение, направление и передвижения устройства, можно:

- использовать компас и акселерометр для определения скорости и направления; сочетая эти данные с картами, камерой и геолокационными сервисами, вы можете создавать интерфейсы дополненной реальности,

которые отображают информацию о местоположении поверх видеопотока с камеры в режиме реального времени;

- создавать пользовательские интерфейсы, динамически изменяющиеся с учетом положения устройства; Android поддерживает управление ориентацией экрана, когда устройство переходит из портретного режима в альбомный и наоборот;
- отслеживать быстрое ускорение — падение или бросок устройства;
- измерять передвижения или вибрацию, например, вы можете создать приложение, которое позволяет заблокировать устройство; если в таком состоянии будет обнаружено какое-то движение, программа может послать уведомление SMS, в котором будет указываться текущее местоположение;
- создавать элементы управления, которые используют физические жесты и движения как средство ввода.

Вы всегда должны проверять наличие всех необходимых датчиков и стремиться к тому, чтобы в случае их недоступности ваше приложение вело себя адекватно.

Знакомство с акселерометром

Акселерометр используется для измерения ускорения. Его иногда называют датчиком силы притяжения.

ПРИМЕЧАНИЕ

Акселерометры часто выступают в качестве датчиков силы притяжения, так как они не могут определить, чем вызвано ускорение — движением или гравитацией. В результате этого в состоянии покоя акселерометр будет указывать на ускорение по оси Z (вверх/вниз), равное 9,8м/с^2 (это значение доступно в виде константы SensorManager.STANDARD_GRAVITY).

Ускорение — это производная скорости по времени, поэтому акселерометр определяет, насколько быстро изменяется скорость устройства в заданном направлении. Используя этот датчик, вы можете обнаруживать движение и, что более полезно, изменение его скорости.

ВНИМАНИЕ

Акселерометр не измеряет скорость как таковую, поэтому вы не можете получить скорость движения, основываясь на единичном замере. Вместо этого необходимо учитывать изменения ускорения на протяжении какого-то отрезка времени.

На практике, скорее всего, вас будет интересовать ускорение относительно состояния покоя или быстрого движения (сопровождаемого рез-

кими изменениями в ускорении); такое ускорение может возникать при использовании жестов для ввода данных. В первом случае вам потребуется откалибровать устройство, вычислив начальные показатели ориентации и ускорения, чтобы учитывать их влияние при вычислении будущих результатов.

Обнаружение изменений ускорения

Ускорение может быть измерено в трех направлениях: по оси ординат, по оси абсцисс, а также по вертикальной оси. SensorManager сообщает об изменениях в показаниях акселерометра по всем трем направлениям.

Значения, переданные через свойство values объекта SensorEvent, отражают боковое, продольное и вертикальное ускорение (именно в таком порядке).

Рисунок 14.1 иллюстрирует нанесение трех направляющих осей на устройство, находящееся в состоянии покоя, которое в интерпретации SensorManager наступает тогда, когда устройство лежит на плоской поверхности экраном вверх и находится при этом в портретном режиме.

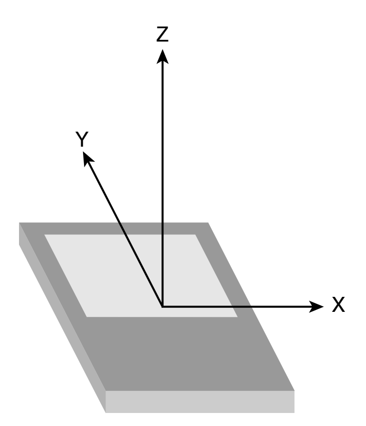


Рис. 14.1

- **Ось X** (ординаты). Боковое (влево или вправо) ускорение, положительные значения которого свидетельствуют о движении в направлении правой части устройства, а отрицательные — в направлении левой его части. Например, положительное ускорение по оси X будет, если устройство, лежа экраном вверх, повернется вправо (не отрывая крышку от поверхности).
- **Ось Y** (абсциссы). Ускорение вперед или назад. В первом случае показатели будут больше нуля. Разместив устройство таким образом, как в предыдущем пункте, и подвинув в сторону его верхнюю часть, вы создадите положительное продольное ускорение.

- **Ось Z** (аппликаты). Ускорение вверх или вниз. В первом случае при подъеме устройства значения будут положительными. В состоянии покоя вертикальное ускорение равно 9,8 м/с^2 (вследствие силы тяжести).

Как уже говорилось ранее, изменения ускорения отслеживаются посредством SensorEventListener. Зарегистрируйте реализацию этого интерфейса с помощью SensorManager, используя объект Sensor с типом Sensor.TYPE_ACCELEROMETER, чтобы запрашивать обновления для акселерометра. В листинге 14.2 регистрируется акселерометр по умолчанию, используя стандартную частоту обновлений.

Листинг 14.2. Отслеживание изменений в показаниях акселерометра по умолчанию

```
SensorManager sm = (SensorManager)getSystemService(Context.SENSOR_
SERVICE);
int sensorType = Sensor.TYPE_ACCELEROMETER;
sm.registerListener(mySensorEventListener,
                    sm.getDefaultSensor(sensorType),
                    SensorManager.SENSOR_DELAY_NORMAL);
```

Интерфейс SensorEventListener должен содержать реализацию обработчика onSensorChanged, который будет срабатывать при измерении ускорения в произвольном направлении.

Метод onSensorChanged получает объект SensorEvent, содержащий массив значений типа float, представляющий собой показатели ускорения по всем трем осям. Основываясь на состоянии покоя, когда устройство лежит на задней крышке в портретном режиме, первый элемент означает боковое ускорение, второй — продольное, третий — вертикальное. Процесс извлечения этих показателей показан в расширении к листингу 14.2:

```
final SensorEventListener mySensorEventListener = new
SensorEventListener() {
  public void onSensorChanged(SensorEvent sensorEvent) {
    if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
      float xAxis_lateralA = sensorEvent.values[0];
      float yAxis_longitudinalA = sensorEvent.values[1];
      float zAxis_verticalA = sensorEvent.values[2];
      // TODO Использовать полученное ускорение в своей программе.
    }
  }
};
```

Создание приложения для измерения перегрузки

Вы можете создать простой инструмент для измерения перегрузки, суммируя ускорение по всем трем направлениям и сравнивая полученный результат с ускорением свободного падения. В следующем примере вы разработаете простой прибор для измерения перегрузки, используя

акселерометр для определения ускорения, которое испытывает данное устройство.

Благодаря силе земного притяжения ускорение устройства в состоянии покоя равняется 9,8 м/с^2 и направлено к центру Земли. В этом примере[1] вы определите влияние силы притяжения с помощью константы SensorManager. STANDARD_GRAVITY.

1. Начните с создания нового проекта Forceometer с одноименной Активностью. Отредактируйте ресурс разметки `main.xml`, чтобы отображать по центру две строки с жирным текстом, который будет содержать текущую и максимальную зафиксированные перегрузки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView android:id="@+id/acceleration"
    android:gravity="center"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textStyle="bold"
    android:textSize="32sp"
    android:text="CENTER"
    android:editable="false"
    android:singleLine="true"
    android:layout_margin="10px"/>
  />
  <TextView android:id="@+id/maxAcceleration"
    android:gravity="center"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textStyle="bold"
    android:textSize="40sp"
    android:text="CENTER"
    android:editable="false"
    android:singleLine="true"
    android:layout_margin="10px"/>
  />
</LinearLayout>
```

2. Внутри Активности Forceometer создайте поля для хранения ссылок на оба элемента TextView и на объект SensorManager. Кроме того, сделайте поле для записи последнего максимального зафиксированного значения ускорения:

```
SensorManager sensorManager;
TextView accelerationTextView;
TextView maxAccelerationTextView;
float currentAcceleration = 0;
float maxAcceleration = 0;
```

[1] Все фрагменты кода в этом примере — часть проекта G-Forceometer из главы 14, их можно загрузить с сайта Wrox.com.

3. Создайте реализацию SensorEventListener, в которой суммируются ускорения по всем трем осям и вычитается ускорение, вызванное силой земного притяжения. Ваша реализация должна обновлять значения текущего и максимального ускорений каждый раз, когда будут фиксироваться изменения в показаниях акселерометра:

```
private final SensorEventListener sensorEventListener = new
SensorEventListener() {
  double calibration = SensorManager.STANDARD_GRAVITY;

  public void onAccuracyChanged(Sensor sensor, int accuracy) { }

  public void onSensorChanged(SensorEvent event) {
    double x = event.values[0];
    double y = event.values[1];
    double z = event.values[2];

    double a = Math.round(Math.sqrt(Math.pow(x, 2) +
                                    Math.pow(y, 2) +
                                    Math.pow(z, 2)));
    currentAcceleration = Math.abs((float)(a-calibration));

    if (currentAcceleration > maxAcceleration)
      maxAcceleration = currentAcceleration;
  }
};
```

4. Обновите метод onCreate, чтобы с помощью SensorManager зарегистрировать новый объект SensorEventListener для фиксации изменений в показаниях акселерометра. Воспользуйтесь возможностью получить ссылки на оба элемента TextView:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  accelerationTextView = (TextView)findViewById(R.id.acceleration);
  maxAccelerationTextView = (TextView)findViewById(R.id.maxAcceleration);
  sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);

  Sensor accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_
ACCELEROMETER);
  sensorManager.registerListener(sensorEventListener,
                                 accelerometer,
                                 SensorManager.SENSOR_DELAY_FASTEST);
}
```

5. Акселерометр может иметь высокую чувствительность, поэтому обновление элементов интерфейса при каждом изменении ускорения способно оказаться очень затратным. Чтобы этого избежать, создайте новый метод updateGUI, который синхронизируется с графическим потоком в зависимости от срабатывания таймера, прежде чем обновлять элементы TextView:

```
private void updateGUI() {
  runOnUiThread(new Runnable() {
    public void run() {
      String currentG = currentAcceleration/SensorManager.STANDARD_GRAVITY
                        + "Gs";
      accelerationTextView.setText(currentG);
      accelerationTextView.invalidate();
      String maxG = maxAcceleration/SensorManager.STANDARD_GRAVITY + "Gs";
      maxAccelerationTextView.setText(maxG);
      maxAccelerationTextView.invalidate();
    }
  });
};
```

6. В завершение обновите метод onCreate, запуская таймер для обновления пользовательского интерфейса каждые 100 мс:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  accelerationTextView = (TextView)findViewById(R.id.acceleration);
  maxAccelerationTextView = (TextView)findViewById(R.id.maxAcceleration);
  sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);

  Sensor accelerometer =
    sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  sensorManager.registerListener(sensorEventListener,
                                 accelerometer,
                                 SensorManager.SENSOR_DELAY_FASTEST);

  Timer updateTimer = new Timer("gForceUpdate");
  updateTimer.scheduleAtFixedRate(new TimerTask() {
    public void run() {
      updateGUI();
    }
  }, 0, 100);
}
```

Закончив писать код, вы явно захотите его проверить. Лучше всего данное приложение было бы тестировать на борту истребителя F-16, выполняющего фигуры высшего пилотажа над Атлантикой. Но подобные эксперименты, как известно, иногда плохо кончаются, поэтому за неимением боевого самолета можете поэкспериментировать с бегом или ездой по району.

Учитывая тот факт, что наблюдать за телефоном во время вождения, езды на велосипеде или полета не очень удобно, вы могли бы подумать, как еще улучшить свое приложение, прежде чем пользоваться им.

Добавьте функции виброзвонка или плеера (чтобы телефон вибрировал или издавал звуковые сигналы, пропорциональные текущему ускорению) или просто записывайте все изменения, чтобы просмотреть их позже.

Определение положения в пространстве

Датчик ориентации — это комбинация датчика магнитного поля, выполняющего роль электронного компаса, и акселерометра, измеряющего наклон и вращение.

Если вы знакомы с тригонометрией, значит есть навыки, необходимые для вычисления положения устройства относительно всех трех осей, основываясь на показаниях акселерометра и датчика магнитного поля. Однако если вы питаете к тригонометрии те же чувства, что и я, то обрадуетесь, узнав, что Android сделает все вычисления сам.

По сути, Android предлагает два способа определения положения устройства. Вы можете сделать запрос напрямую к датчику ориентации или получить необходимые данные с помощью акселерометра и датчика магнитного поля. Второй вариант более медленный, но имеет преимущества — повышенную точность и возможность изменять систему отсчета при определении положения в пространстве. В следующих разделах описываются оба подхода.

При использовании стандартной системы отсчета положение устройства вычисляется в трех измерениях, как показано на рис. 14.2. Как и в случае с акселерометром, устройство рассматривается в состоянии покоя, лежа экраном вверх на плоской поверхности.

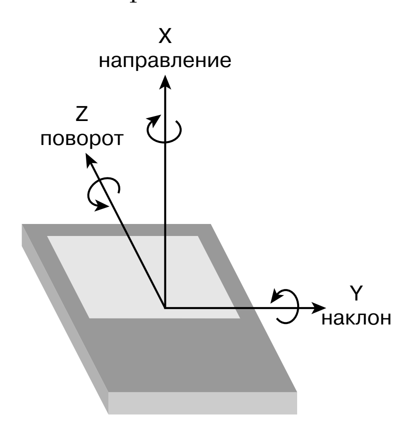


Рис. 14.2

- **Ось X** (направление). Направление устройства при движении вокруг оси X; 0°/360° — север, 90° — восток, 180° — юг, 270° — запад.
- **Ось Y** (наклон). Угол наклона устройства при вращении относительно оси Y. Он равняется 0°, если устройство лежит на задней крышке, –90° — если расположено вертикально (верхняя часть устройства указывает вверх), 90° — если перевернуто, 180°/–180° — если экраном вниз.

- **Ось Z** (поворот). Поворот писывает боковой наклон устройства по оси Z между –90° и 90°. При 0° устройство лежит на задней крышке, при –90° экран повернут влево, при 90° — вправо.

Определение положения в пространстве с помощью датчика ориентации

Самый простой способ отслеживать положение устройства в пространстве — применение отдельного датчика ориентации. Создайте и зарегистрируйте SensorEventListener с помощью объекта SensorManager, используя датчик ориентации по умолчанию, как показано в листинге 14.3.

Листинг 14.3. Определение положения в пространстве с помощью датчика ориентации

```
SensorManager sm = (SensorManager)getSystemService(Context.SENSOR_
SERVICE);
int sensorType = Sensor.TYPE_ORIENTATION;
sm.registerListener(myOrientationListener,
                    sm.getDefaultSensor(sensorType),
                    SensorManager.SENSOR_DELAY_NORMAL);
```

Когда положение устройства меняется, в вашей реализации интерфейса SensorEventListener срабатывает обработчик onSensorChanged. Параметр SensorEvent включает в себя массив значений типа float, описывающий положение устройства по трем осям. Первый элемент этого массива — направление, второй — наклон, третий — поворот.

```
final SensorEventListener myOrientationListener = new
SensorEventListener() {
  public void onSensorChanged(SensorEvent sensorEvent) {
    if (sensorEvent.sensor.getType() == Sensor.TYPE_ORIENTATION) {
      float headingAngle = sensorEvent.values[0];
      float pitchAngle = sensorEvent.values[1];
      float rollAngle = sensorEvent.values[2];

      // TODO Использовать изменение положения в своей программе.
    }
  }

  public void onAccuracyChanged(Sensor sensor, int accuracy) {}
};
```

Вычисление положения с помощью акселерометра и датчика магнитного поля

Лучший способ определить положение устройства в пространстве — вычислить его, используя напрямую показания акселерометра и датчика магнитного поля.

Данный подход позволяет вам изменять систему отсчета для переназначения осей X, Y и Z, чтобы корректировать ожидаемую ориентацию устройства.

При таком подходе используются сразу два датчика — акселерометр и датчик магнитного поля, поэтому вам необходимо создать и зарегистрировать две реализации SensorEventListener. Внутри методов onSensorChanged для каждой из них записывайте свойство values, получаемое из двух разных полей, как показано в листинге 14.4.

Листинг 14.4. Определение положения в пространстве с помощью акселерометра и датчика магнитного поля

```
float[] accelerometerValues;
float[] magneticFieldValues;

final SensorEventListener myAccelerometerListener = new
SensorEventListener() {
  public void onSensorChanged(SensorEvent sensorEvent) {
    if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
      accelerometerValues = sensorEvent.values;
  }

  public void onAccuracyChanged(Sensor sensor, int accuracy) {}
};

final SensorEventListener myMagneticFieldListener = new
SensorEventListener() {
  public void onSensorChanged(SensorEvent sensorEvent) {
    if (sensorEvent.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
      magneticFieldValues = sensorEvent.values;
  }

  public void onAccuracyChanged(Sensor sensor, int accuracy) {}
};
```

Зарегистрируйте обе реализации SensorEventListener с помощью объекта SensorManager, как показано в следующем дополнении к листингу 14.4. В этом фрагменте используются аппаратные датчики по умолчанию и частота обновления SENSOR_DELAY_UI:

```
SensorManager sm = (SensorManager)getSystemService(Context.SENSOR_
SERVICE);
Sensor aSensor = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
Sensor mfSensor = sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

sm.registerListener(myAccelerometerListener,
                    aSensor,
                    SensorManager.SENSOR_DELAY_UI);

sm.registerListener(myMagneticFieldListener,
                    mfSensor,
                    SensorManager.SENSOR_DELAY_UI);
```

Чтобы вычислить текущее положение в пространстве с помощью показаний этих датчиков, используйте методы getRotationMatrix и getOrientation

из объекта SensorManager, как показано ниже. Обратите внимание на то, что getOrientation возвращает радианы, а не градусы.

```
float[] values = new float[3];
float[] R = new float[9];
SensorManager.getRotationMatrix(R, null,
                                accelerometerValues,
                                magneticFieldValues);
SensorManager.getOrientation(R, values);

// Преобразуйте радианы в градусы.
values[0] = (float) Math.toDegrees(values[0]);
values[1] = (float) Math.toDegrees(values[1]);
values[2] = (float) Math.toDegrees(values[2]);
```

Переопределение системы отсчета

Чтобы вычислить положение устройства в пространстве с помощью системы отсчета, отличной от стандартной (описанной выше), используйте метод remapCoordinateSystem из объекта SensorManager.

Ранее в этой главе стандартная система отсчета описывалась таким образом, что устройство должно было лежать на плоской поверхности экраном вверх. Подобный подход позволяет вам изменять систему координат для вычисления положения в пространстве. Таким образом, положение, при котором устройство расположено вертикально, можно обозначить как состояние покоя.

Метод remapCoordinateSystem принимает четыре параметра:

- исходная матрица поворота, получаемая методом getRotationMatrix, как описывалось ранее;
- переменная, используемая для хранения конечной (преобразованной) матрицы поворота;
- переопределенная ось X;
- переопределенная ось Y.

Два последних параметра используются для указания новой системы отсчета. Эти значения определяют новые оси X и Y, смещенные относительно стандартных. Класс SensorManager предоставляет набор констант, позволяющих указывать значения для осей: AXIS_X, AXIS_Y, AXIS_Z, AXIS_MINUS_X, AXIS_MINUS_Y и AXIS_MINUS_Z.

В листинге 14.5 показывается, как переопределить систему отсчета таким образом, чтобы состояние покоя наступало, когда устройство расположено вертикально в портретном режиме, экран повернут к пользователю (рис.14.3).

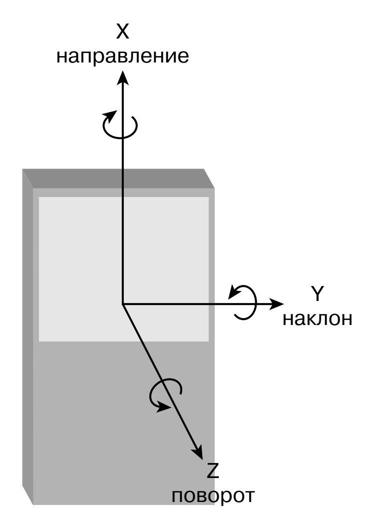


Рис. 14.3

Листинг 14.5. Переопределение системы отсчета

```
SensorManager.getRotationMatrix(R, null, aValues, mValues);

float[] outR = new float[9];
SensorManager.remapCoordinateSystem(R,
                                    SensorManager.AXIS_X,
                                    SensorManager.AXIS_Z,
                                    outR);
SensorManager.getOrientation(outR, values);

// Преобразуйте радианы в градусы.
values[0] = (float) Math.toDegrees(values[0]);
values[1] = (float) Math.toDegrees(values[1]);
values[2] = (float) Math.toDegrees(values[2]);
```

Создание компаса и автогоризонта

В главе 4 вы разработали простой элемент CompassView, чтобы поэкспериментировать с созданием собственных графических элементов. В этом примере[1] вы расширите возможности CompassView, сначала научив его показывать параметры pitch и roll, а затем и положение устройства в пространстве.

1. Откройте проект Compass, созданный при изучении главы 4. Нужно внести некоторые изменения в элемент CompassView, а также в Активность Compass, в которой он размещен. Чтобы обеспечить максимальную независимость логики от представления, элемент CompassView

[1] Все фрагменты кода в этом примере — часть проекта Artificial Horizon из главы 14, их можно загрузить с сайта Wrox.com.

не будет взаимодействовать с датчиками напрямую. Вместо этого он будет обновляться из Активности. Начните с добавления в класс CompassView полей и геттеров/сеттеров для параметров pitch и roll.

```
float pitch = 0;
float roll = 0;

public float getPitch() {
  return pitch;
}
public void setPitch(float pitch) {
  this.pitch = pitch;
}

public float getRoll() {
  return roll;
}
public void setRoll(float roll) {
  this.roll = roll;
}
```

2. Обновите метод onDraw, нарисовав с его помощью две окружности, которые будут использоваться для отображения значений pitch и roll.

```
@Override
protected void onDraw(Canvas canvas) {

[ ... Ранее написанный код ... ]
```

2.1. Создайте новую окружность, закрашенную наполовину, которая вращается в соответствии с боковым наклоном (roll).

```
RectF rollOval = new RectF((mMeasuredWidth/3)-mMeasuredWidth/7,
                           (mMeasuredHeight/2)-mMeasuredWidth/7,
                           (mMeasuredWidth/3)+mMeasuredWidth/7,
                           (mMeasuredHeight/2)+mMeasuredWidth/7
                          );
markerPaint.setStyle(Paint.Style.STROKE);
canvas.drawOval(rollOval, markerPaint);
markerPaint.setStyle(Paint.Style.FILL);
canvas.save();
canvas.rotate(roll, mMeasuredWidth/3, mMeasuredHeight/2);
canvas.drawArc(rollOval, 0, 180, false, markerPaint);

canvas.restore();
```

2.2. Создайте еще одну окружность, которая в состоянии покоя закрашена наполовину и, в зависимости от изменений параметра pitch, может быть как пустой, так и полностью закрашенной:

```
  RectF pitchOval = new RectF((2*mMeasuredWidth/3)-mMeasuredWidth/7,
                              (mMeasuredHeight/2)-mMeasuredWidth/7,
                              (2*mMeasuredWidth/3)+mMeasuredWidth/7,
                              (mMeasuredHeight/2)+mMeasuredWidth/7
                             );
```

```
  markerPaint.setStyle(Paint.Style.STROKE);
  canvas.drawOval(pitchOval, markerPaint);
  markerPaint.setStyle(Paint.Style.FILL);
  canvas.drawArc(pitchOval, 0-pitch/2, 180+(pitch), false, markerPaint);
  markerPaint.setStyle(Paint.Style.STROKE);
}
```

3. На этом процесс внесения изменений в класс CompassView можно считать завершенным. Если теперь запустить приложение, оно должно выглядеть, как на рис. 14.4.

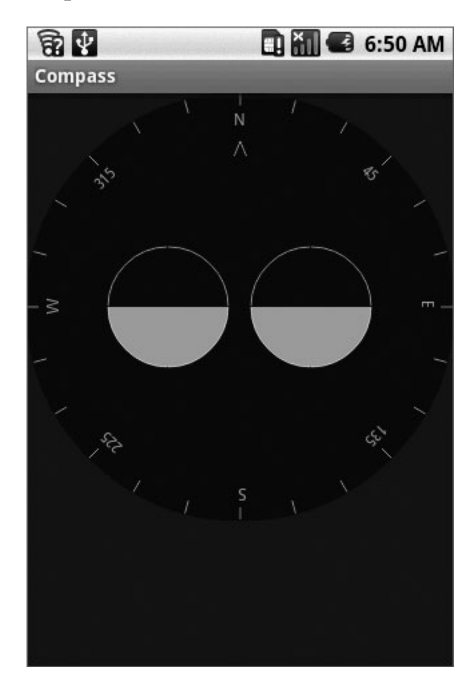


Рис. 14.4.

4. Теперь перейдите к Активности Compass. Для отслеживания положения устройства в пространстве с помощью датчика магнитного поля и акселерометра используйте объект SensorManager. Сперва добавьте поля для хранения последних зафиксированных показаний этих датчиков, а также ссылки на объекты CompassView и SensorManager.

```
float[] aValues = new float[3];
float[] mValues = new float[3];
CompassView compassView;
SensorManager sensorManager;
```

5. Создайте новый метод updateOrientation, который использует актуальный азимут, а также значения pitch и roll для обновления элемента CompassView.

```
private void updateOrientation(float[] values) {
  if (compassView!= null) {
    compassView.setBearing(values[0]);
    compassView.setPitch(values[1]);
    compassView.setRoll(-values[2]);
    compassView.invalidate();
  }
}
```

6. Обновите метод onCreate, чтобы получить ссылки на объекты CompassView и SensorManager, инициализировав азимут и значения pitch и roll.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  compassView = (CompassView)this.findViewById(R.id.compassView);
  sensorManager = (SensorManager)getSystemService(Context.SENSOR_
SERVICE);
  updateOrientation(new float[] {0, 0, 0});
}
```

7. Создайте новый метод calculateOrientation для вычисления положения устройства в пространстве, используя поля, хранящие последние записанные показания акселерометра и датчика магнитного поля.

```
private float[] calculateOrientation() {
  float[] values = new float[3];
  float[] R = new float[9];

  SensorManager.getRotationMatrix(R, null, aValues, mValues);
  SensorManager.getOrientation(R, values);

  // Преобразуйте радианы в градусы.
  values[0] = (float) Math.toDegrees(values[0]);
  values[1] = (float) Math.toDegrees(values[1]);
  values[2] = (float) Math.toDegrees(values[2]);

  return values;
}
```

8. Реализуйте интерфейс SensorEventListener в виде поля. Внутри обработчика onSensorChanged нужно проверить тип датчика и обновить поля, хранящие последние показания акселерометра или датчика магнитного поля, прежде чем вызвать метод updateOrientation с помощью calculateOrientation.

```
private final SensorEventListener sensorEventListener = new
SensorEventListener() {

  public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
      aValues = event.values;
```

```
    if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
      mValues = event.values;

    updateOrientation(calculateOrientation());
  }

  public void onAccuracyChanged(Sensor sensor, int accuracy) {}
};
```

9. Теперь переопределите обработчики onResume и onStop, чтобы регистрировать и отменять регистрацию SensorEventListener, когда Активность становится видимой или скрывается.

```
@Override
protected void onResume()
{
  super.onResume();

  Sensor accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_
ACCELEROMETER);
  Sensor magField = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_
FIELD);

  sensorManager.registerListener(sensorEventListener,
                                 accelerometer,
                                 SensorManager.SENSOR_DELAY_FASTEST);
  sensorManager.registerListener(sensorEventListener,
                                 magField,
                                 SensorManager.SENSOR_DELAY_FASTEST);
}

@Override
protected void onStop()
{
  sensorManager.unregisterListener(sensorEventListener);
  super.onStop();
}
```

Запустив приложение, вы должны увидеть три индикатора, которые динамически обновляются при изменении положения устройства в пространстве.

10. Автогоризонт более полезен в вертикальном положении. Для этого необходимо изменить систему отсчета автогоризонта, обновив метод calculateOrientation, в котором должна преобразовываться система координат.

```
private float[] calculateOrientation() {
  float[] values = new float[3];
  float[] R = new float[9];
  float[] outR = new float[9];

  SensorManager.getRotationMatrix(R, null, aValues, mValues);
  SensorManager.remapCoordinateSystem(R,
                                      SensorManager.AXIS_X,
```

```
                                          SensorManager.AXIS_Z,
                                          outR);

    SensorManager.getOrientation(outR, values);

    // Преобразуйте радианы в градусы.
    values[0] = (float) Math.toDegrees(values[0]);
    values[1] = (float) Math.toDegrees(values[1]);
    values[2] = (float) Math.toDegrees(values[2]);

    return values;
  }
```

Управление вибрацией устройства

В главе 9 вы научились создавать уведомления, которые могли использовать вибрацию для улучшения обратного отклика. В некоторых случаях вам может понадобиться использовать вибрацию безотносительно уведомлений. Включение вибрации в устройстве — отличный способ привлечь внимание пользователя посредством тактильного воздействия, который получил широкое распространение в играх.

Чтобы управлять вибрацией устройства, ваше приложение должно иметь полномочие VIBRATE. Добавьте его в манифест, как показано в следующем фрагменте кода:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

Управляет вибрацией устройства Сервис Vibrator, доступ к которому можно получить через метод getSystemService, как показано в листинге 14.6.

Листинг 14.6. Управление вибрацией устройства

```
String vibratorService = Context.VIBRATOR_SERVICE;
Vibrator vibrator = (Vibrator)getSystemService(vibratorService);
```

Вызовите метод vibrate, чтобы заставить устройство вибрировать; вы можете передать в качестве параметров продолжительность вибрации или шаблон ее воспроизведения (последовательность вибрация/пауза), а также необязательный индекс, указывающий на позицию, с которой следует начать повторение этого шаблона. Оба эти подхода показаны в следующем дополнении к листингу 14.6:

```
long[] pattern = {1000, 2000, 4000, 8000, 16000 };
vibrator.vibrate(pattern, 0); // Выполнить шаблон воспроизведения вибрации.
vibrator.vibrate(1000); // Вибрировать на протяжении одной секунды.
```

Чтобы отменить вибрацию, вызовите метод cancel. При выходе из приложения все вибрации, инициированные им, будут отменены.

Резюме

В этой главе вы научились использовать объект SensorManager, позволяя своим приложениям реагировать на изменения окружающего мира. Познакомились с датчиками, доступными на платформе Android, и узнали, как отслеживать и интерпретировать их показания с помощью SensorEventListener.

Затем вы более подробно изучили акселерометр, датчики ориентации и магнитного поля; использовали объекты Sensor для определения ускорения и положения устройства в пространстве. При этом создали измеритель перегрузок и автогоризонт.

Вы также узнали:

- какие датчики доступны для приложений, выполняющихся на платформе Android;
- как изменить систему отсчета при определении положения устройства в пространстве;
- состав и назначение объектов SensorEvent, возвращаемых каждым датчиком;
- как использовать вибрацию устройства в качестве физического отклика для событий внутри приложения.

В последней главе описаны некоторые продвинутые возможности платформы Android. Вы узнаете больше о безопасности, о том, как использовать Сервис WakeLock и язык AIDL для облегчения межпроцессного взаимодействия, о пользовательском интерфейсе и графических возможностях. Познакомитесь с библиотеками Android для преобразования текста в речь, изучите продвинутые механизмы для поддержки анимации и рисования с помощью объекта Canvas. И в завершение мы поговорим об объекте SurfaceView и опишем возможности ввода информации через сенсорный экран.

Глава 15
ПРОДВИНУТОЕ ПРОГРАММИРОВАНИЕ ПОД ANDROID

Содержание главы

- Обеспечение безопасности в Android с помощью системы полномочий.
- Использование объектов WakeLock.
- Библиотеки для преобразования текста в речь.
- Межпроцессное взаимодействие (IPC) с помощью интерфейсов AIDL и Parcelable.
- Создание анимации пошаговым методом и путем расчета промежуточных кадров.
- Продвинутое рисование с помощью Canvas.
- Использование объекта SurfaceView.
- Отслеживание движений трекбола, прикосновений к сенсорному экрану и нажатий на клавиши.

В этой главе вы вернетесь к некоторым возможностям, рассмотренным ранее, а также исследуете другие темы, заслуживающие более пристального внимания.

В первых семи главах вы изучили основы создания мобильных приложений для устройств под управлением Android. В главах 8–14 познакомились с некоторыми дополнительными и более мощными программными интерфейсами, включая геолокационные сервисы, карты, Bluetooth, процесс слежения и контроля за аппаратным обеспечением.

Данная глава начинается с углубленного изучения безопасности, в частности, принципов работы полномочий; вы узнаете, как использовать полномочия, чтобы обезопасить собственные приложения.

Далее рассмотрите класс WakeLock и библиотеки для преобразования текста в речь, после на очереди язык описания интерфейсов в Android (Android Interface Definition Language или AIDL). Вы будете использовать AIDL

для создания насыщенных программных интерфейсов, которые поддерживают полноценное объектно-ориентированное межпроцессное взаимодействие (IPC) между приложениями, работающими в разных процессах.

Вы более детально изучите богатый инструментарий для создания пользовательских интерфейсов. Вы узнаете, как создавать анимацию для отдельных Представлений и их групп с помощью расчета промежуточных кадров, а также пошаговую мультипликационную анимацию.

Затем перейдете к углубленному изучению возможностей, предоставляемых растровым графическим движком в Android. Познакомитесь с набором доступных для рисования примитивов, прежде чем обратиться к расширенным возможностям объекта Paint. Узнаете, как использовать прозрачность, создавать градиенты с помощью шейдеров и смешивать растровые кисти. Изучите маски, цветовые фильтры, объекты PathEffect и возможности различных режимов преобразования.

В завершение углубитесь в разработку и применение Представлений с более сложным пользовательским интерфейсом, узнаете, как с помощью SurfaceView создавать трехмерные интерактивные элементы управления, поддерживающие высокую частоту смены кадров, научитесь использовать сенсорный экран, трекбол и аппаратные клавиши для создания интуитивно понятных возможностей ввода для своих графических интерфейсов.

Paranoid Android

Система безопасности в Android во многом основывается на ядре Linux. Ресурсы изолированы в рамках родительских приложений, что делает их недоступными извне. Android предоставляет Широковещательные намерения, Сервисы и Источники данных, чтобы появилась возможность выходить за строгие рамки процессов, используя механизм полномочий для поддержания безопасности на уровне приложения.

Вы уже использовали механизм полномочий для получения доступа к стандартным системным компонентам, в частности к геолокационным сервисам и к Источнику данных, содержащему информацию о контактах. Делалось это с помощью тегов `<uses-permission>` в манифесте приложения.

В следующих разделах более подробно рассмотрены доступные на платформе Android механизмы обеспечения безопасности. Чтобы получить о них исчерпывающее представление, можете ознакомиться с документацией, размещенной на сайте для разработчиков по адресу developer.android.com/guide/topics/security/security.html.

Безопасность на уровне ядра Linux

Все пакеты в Android имеют собственный уникальный пользовательский идентификатор (UID) на уровне ядра Linux. Благодаря этому процессы

и ресурсы, созданные ими, изолируются и не могут повлиять на другие приложения (равно как и сами подвергнуться влиянию).

Учитывая такую систему безопасности, работающую на уровне ядра, необходимо предпринимать дополнительные действия, чтобы связываться с другими приложениями. Делается это с помощью Источников данных, Широковещательных намерений и интерфейсов AIDL. Каждый из этих механизмов открывает своеобразный тоннель, через который приложения могут обмениваться информацией. При этом система полномочий выступает в качестве «пограничников», контролирующих потоки данных на обоих концах.

Знакомство с системой полномочий

Полномочия — это механизм безопасности, функционирующий на прикладном уровне и позволяющий ограничивать доступ к компонентам приложения. Они нужны для защиты от вредоносных программ, которые могут нанести ущерб, получая доступ к конфеденциальной информации или используя (возможно, несанкционированно) чрезмерное количество аппаратных ресурсов или возможностей внешних каналов связи.

Как вы уже могли заметить в предыдущих главах, многие стандартные компоненты в Android требуют наличия полномочий. Стандартные строки для описания полномочий, используемых системными Активностями и Сервисами в Android, можно найти в классе android.Manifest.permission в виде статических констант.

Чтобы использовать компоненты, защищенные данным механизмом, необходимо добавить теги `<uses-permission>` в манифест своего приложения, указывая требуемые полномочия.

После установки приложения полномочия, запрашиваемые в его манифесте, анализируются и предоставляются (или отвергаются) с помощью доверенных сертификатов в зависимости от решения пользователя.

В отличие от многих других мобильных платформ в Android все полномочия проверяются во время установки. Если приложение установлено, пользователя больше не будут спрашивать о пересмотре этих полномочий.

Объявление и обеспечение полномочий

Прежде чем назначить полномочие для компонента приложения, необходимо объявить его в манифесте, используя тег `<permission>`, как показано в листинге 15.1.

Листинг 15.1. Объявление нового полномочия

```
<permission
  android:name="com.paad.DETONATE_DEVICE"
  android:protectionLevel="dangerous"
  android:label="Self Destruct"
  android:description="@string/detonate_description">
</permission>
```

Внутри данного тега вы можете задать уровень доступа, предоставляемого полномочием (normal, dangerous, signature, signatureOrSystem), метку, а также внешний ресурс, содержащий описание рисков, которыми сопровождается выдача этого полномочия.

Чтобы задать требования к полномочиям для собственных программных компонентов, используйте атрибут `permission` в манифесте приложения. Такие ограничения могут распространяться на все ваше приложение целиком, хотя наиболее полезны в связке с внешними программными интерфейсами.

- **Активности.** Добавьте полномочие, чтобы ограничить возможность запуска вашей Активности другими приложениями.
- **Приемники широковещательных намерений.** Следите за тем, какое приложение может слать Широковещательные намерения в ваш Приемник.
- **Источники данных.** Ограничивайте доступ к Источникам данных на чтение/запись.
- **Сервисы.** Ограничивайте возможность запускать Сервис или связываться с ним из других приложений.

В каждом из этих случаев вы можете добавить атрибут `permission` к компоненту приложения в манифесте, указав строку, описывающую необходимое полномочие; таким образом, отрегулируете доступ к нему. В листинге 15.2 показан фрагмент манифеста, в котором описывается требование полномочия, заданного в листинге 15.1. Без выполнения этого требования Активность не запустится.

Листинг 15.2. Запрос полномочий у Активности

```
<activity
  android:name=".MyActivity"
  android:label="@string/app_name"
  android:permission="com.paad.DETONATE_DEVICE">
</activity>
```

С помощью Источников данных можно задавать атрибуты для readPermission и writePermission, благодаря чему вы можете более тонко контролировать доступ к чтению и записи.

Обеспечение полномочий для Приемников широковещательных намерений

Как и в случае с требованием полномочий от Намерений, принимаемых Широковещательными приемниками, вы также можете добавить подобные требования в сами Намерения, которые транслируете.

При вызове метода sendIntent можно указать строку с полномочием, без наличия которого Приемник не сможет принять данное:

```
sendBroadcast(myIntent, REQUIRED_PERMISSION);
```

Использование объекта WakeLock

С целью увеличить длительность автономной работы устройства на платформе Android со временем постепенно уменьшают яркость подсветки, выключают экран — и в итоге отключают центральный процессор. WakeLock — объект, предоставляемый Сервисом PowerManager и позволяющий вашим приложениям управлять питанием устройства.

Объект WakeLock может использоваться для поддержания ЦПУ в рабочем режиме, предотвращения затухания (речь идет о случаях, когда пользователь, вероятно, не очень активно взаимодействует с приложением и смотрит при этом на экран, например просматривая видео) и выключения экрана, отключения подсветки клавиатуры.

ВНИМАНИЕ

Применение объектов WakeLock способно значительно повлиять на использование вашим приложением ресурсов батареи. На практике старайтесь задействовать WakeLock только в случае крайней необходимости и на протяжении короткого промежутка времени.

WakeLock также может запрещать устройству входить в режим ожидания, пока идут какие-то операции. Чаще всего это делается для Сервисов, запускаемых внутри объектов IntentReceiver: они могут принимать Намерения, пока устройство находится в режиме ожидания. Стоит отметить, что в этом случае система будет поддерживать активное состояние устройства до тех пор, пока не завершится выполнение метода onReceive из Приемника намерений.

ПРИМЕЧАНИЕ

Если вы запустите Сервис или передадите Намерение внутри обработчика onReceive, принадлежащего объекту IntentReceiver, вполне возможно, что WakeLock завершит работу еще до выполнения необходимых операций. Чтобы гарантировать своевременный запуск Сервиса, необходимо задать отдельный объект WakeLock.

Чтобы создать экземпляр класса WakeLock, вызовите метод newWakeLock из Сервиса PowerManager, указав один из следующих типов:

- FULL_WAKE_LOCK — поддерживает максимальную установленную яркость экрана, подсветку клавиатуры и рабочий режим ЦПУ;

- SCREEN_BRIGHT_WAKE_LOCK — поддерживает максимальную установленную яркость экрана и рабочий режим ЦПУ;
- SCREEN_DIM_WAKE_LOCK — поддерживает работу экрана (позволяя затухание) и рабочий режим ЦПУ;
- PARTIAL_WAKE_LOCK — поддерживает рабочий режим ЦПУ.

```
PowerManager pm = (PowerManager)getSystemService(Context.POWER_SERVICE);
WakeLock wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
                                          "MyWakeLock");
```

После создания объект WakeLock нужно запустить. При необходимости вы можете задать параметр timeout, чтобы обеспечить максимальную продолжительность работы устройства без перехода в режим ожидания. После того как все необходимые действия завершены, вызовите метод release, чтобы система могла продолжить управление питанием устройства.

В листинге 15.3 показан типичный шаблон создания, запуска и остановки объекта WakeLock.

Листинг 15.3. Использование объекта WakeLock

```
PowerManager pm = (PowerManager)getSystemService(Context.POWER_SERVICE);
WakeLock wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
                                   "MyWakeLock");
wakeLock.acquire();
[ ... Выполнение действий, нуждающихся в сохранении рабочего
режима ЦПУ ... ]
wakeLock.release();
```

Преобразование текста в речь на платформе Android

В Android 1.6 (SDK API level 4) представлен движок для преобразования текста в речь (text to speech, или TTS). Вы можете использовать этот API, чтобы синтезировать речь внутри своих приложений, позволяя им «разговаривать» с пользователями.

Поскольку место для хранения данных на некоторых устройствах под управлением Android ограничено, языковые пакеты не всегда изначально присутствуют в системе. Прежде чем запускать движок TTS, рекомендуется проверить наличие этих языковых библиотек.

Для этого запустите новую Активность с помощью метода startActivityForResult, используя Намерение с действием ACTION_CHECK_TTS_DATA.

```
Intent intent = new Intent(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(intent, TTS_DATA_CHECK);
```

Если соответствующие библиотеки успешно установлены, обработчик onActivityResult получит действие CHECK_VOICE_DATA_PASS.

Если данные для воспроизведения речи недоступны, запустите новую Активность, используя действие ACTION_INSTALL_TTS_DATA из класса TextToSpeech.Engine. Тем самым вы инициируете установку.

Подтвердив наличие всех необходимых данных, нужно создать и инициализировать новый экземпляр класса TextToSpeech. Обратите внимание, что вы не можете использовать объект TextToSpeech, пока не завершится его инициализация. Добавьте в конструктор реализацию интерфейса OnInitListener (показано в листинге 15.4), при ее срабатывании объект готов к использованию.

Листинг 15.4. Инициализация объекта TextToSpeech

```
boolean ttsIsInit = false;
TextToSpeech tts = null;

tts = new TextToSpeech(this, new OnInitListener() {
        public void onInit(int status) {
          if (status == TextToSpeech.SUCCESS) {
            ttsIsInit = true;
            // TODO Воспроизведение речи.
          }
        }
      });
```

Инициализировав объект TextToSpeech, вы можете использовать метод speak, чтобы синтезировать речь с помощью стандартного аудиовыхода.

```
tts.speak("Hello, Android", TextToSpeech.QUEUE_ADD, null);
```

Дополнительный параметр метода speak позволяет выбирать, нужно ли добавлять новый голосовой вывод в уже имеющуюся очередь или начинать воспроизведение немедленно.

Вы можете изменять способ, с помощью которого выводится голос, используя методы setPitch и setSpeechRate. Каждый из них принимает в качестве параметра значение типа float, описывающее высоту и скорость воспроизведения голосового вывода соответственно.

Что еще более важно, вы можете изменять произношение выводимой речи, используя метод setLanguage. Он принимает объект типа Local для задания страны и языка, относящихся к воспроизводимому тексту. Это влиет на то, каким образом произносится текст, позволяет выбрать правильную языковую модель и тип произношения.

Закончив воспроизведение речи, используйте метод stop, чтобы остановить голосовой вывод, и shutdown, чтобы освободить ресурсы движка TTS.

В листинге 15.5 показано, как определить наличие установленной библиотеки TTS, инициализировать новый объект TextToSpeech и использовать его для разговора на британском варианте английского языка.

Листинг 15.5. Преобразование текста в речь

```
private static int TTS_DATA_CHECK = 1;

private TextToSpeech tts = null;
private boolean ttsIsInit = false;

private void initTextToSpeech() {
  Intent intent = new Intent(Engine.ACTION_CHECK_TTS_DATA);
  startActivityForResult(intent, TTS_DATA_CHECK);
}

protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
  if (requestCode == TTS_DATA_CHECK) {
    if (resultCode == Engine.CHECK_VOICE_DATA_PASS) {
      tts = new TextToSpeech(this, new OnInitListener() {
        public void onInit(int status) {
          if (status == TextToSpeech.SUCCESS) {
            ttsIsInit = true;
            if (tts.isLanguageAvailable(Locale.UK) >= 0)
              tts.setLanguage(Locale.UK);
            tts.setPitch(0.8f);
            tts.setSpeechRate(1.1f);
            speak();
          }
        }
      });
    } else {
      Intent installVoice = new Intent(Engine.ACTION_INSTALL_TTS_DATA);
      startActivity(installIntent);
    }
  }
}

private void speak() {
  if (tts != null && ttsIsInit) {
    tts.speak("Hello, Android", TextToSpeech.QUEUE_ADD, null);
  }
}

@Override
public void onDestroy() {
  if (tts != null) {
    tts.stop();
    tts.shutdown();
  }
  super.onDestroy();
}
```

Использование AIDL при межпроцессном взаимодействии Сервисов

Одна из наиболее интересных возможностей Сервисов — запуск независимых фоновых процессов, выполняющих обработку и поиск данных или другие полезные функции для разных приложений.

В главе 9 вы узнали, как создавать Сервисы для своих программ. Теперь пришло время научиться использовать язык описания интерфейсов (Android Interface Definition Language или AIDL) для обеспечения поддержки межпроцессного взаимодействия (IPC) между Сервисами и компонентами приложения. Это позволит вашим Сервисам работать с разными программами, не ограничиваясь рамками единственного процесса.

Чтобы обмениваться объектами между процессами, необходимо разбить их на простые элементы, предоставив тем самым возможность операционной системе выносить их за рамки приложений.

AIDL используется для упрощения кода, предлагающего процессам обмениваться объектами. Этот механизм похож на такие интерфейсы, как COM или Corba, с помощью которых создаются публичные методы внутри Сервисов. Эти методы могут принимать и возвращать объектные параметры, передавая значения между процессами.

Реализация интерфейсов с помощью AIDL

AIDL поддерживает несколько типов данных.

- Простые типы Java (int, boolean, float, char и т. д.).
- Значения String и CharSequence.
- Объекты List (включая обобщенные), где каждый элемент должен иметь поддерживаемый тип. Класс-получатель всегда будет принимать объект List, приведенный к типу ArrayList.
- Объекты Map (обобщенные типы не допускаются), где каждый ключ и элемент должен иметь поддерживаемый тип. Класс-получатель всегда будет принимать объект Map, приведенный к типу HashMap.
- Интерфейсы, порожденные с помощью AIDL (рассматриваются позднее). Всегда нуждается в операторе import.
- Классы, реализующие интерфейс Parcelable (рассматриваются далее). Всегда нуждается в операторе import.

В следующих разделах вы узнаете, как сделать классы своего приложения совместимыми с AIDL с помощью реализации интерфейса Parcelable, рассмотрите объявление интерфейса AIDL и реализацию его внутри Сервиса.

Передача объектов произвольного класса в виде реализаций интерфейса Parcelable

Чтобы передавать нестандартные объекты между процессами, они должны реализовывать интерфейс Parcelable. Это позволит разбивать их на простые типы, хранимые в объекте Parcel, который, в свою очередь, может передаваться от процесса к процессу.

Реализуйте метод writeToParcel, чтобы разбить объект класса на составные части, затем публичное статическое поле Creator (представляющее собой класс Parcelable.Creator), которое создаст новый объект, основанный на переданном экземпляре Parcel.

В листинге 15.6 показан пример использования интерфейса Parcelable в сочетании с классом Quake, который вы ранее создали в своем проекте Earthquake.

Листинг 15.6. Реализация интерфейса Parcelable для класса Quake

```
package com.paad.earthquake;

import java.util.Date;
import android.location.Location;
import android.os.Parcel;
import android.os.Parcelable;

public class Quake implements Parcelable {
  private Date date;
  private String details;
  private Location location;
  private double magnitude;
  private String link;

  public Date getDate() { return date; }
  public String getDetails() { return details; }
  public Location getLocation() { return location; }
  public double getMagnitude() { return magnitude; }
  public String getLink() { return link; }

  public Quake(Date _d, String _det, Location _loc,
               double _mag, String _link) {
    date = _d;
    details = _det;
    location = _loc;
    magnitude = _mag;
    link = _link;
  }

  @Override
  public String toString(){
    SimpleDateFormat sdf = new SimpleDateFormat("HH.mm");
    String dateString = sdf.format(date);
```

```
    return dateString + ":" + magnitude + " " + details;
  }

  private Quake(Parcel in) {
    date.setTime(in.readLong());
    details = in.readString();
    magnitude = in.readDouble();
    Location location = new Location("generated");
    location.setLatitude(in.readDouble());
    location.setLongitude(in.readDouble());
    link= in.readString();
  }

  public void writeToParcel(Parcel out, int flags) {
    out.writeLong(date.getTime());
    out.writeString(details);
    out.writeDouble(magnitude);
    out.writeDouble(location.getLatitude());
    out.writeDouble(location.getLongitude());
    out.writeString(link);
  }

  public static final Parcelable.Creator<Quake> CREATOR =
    new Parcelable.Creator<Quake>() {
      public Quake createFromParcel(Parcel in) {
        return new Quake(in);
      }

      public Quake[] newArray(int size) {
        return new Quake[size];
      }
    };

  public int describeContents() {
    return 0;
  }
}
```

Теперь, когда у вас есть реализация Parcelable, необходимо создать объявление AIDL, чтобы сделать класс доступным при реализации интерфейса AIDL для своего Сервиса.

В листинге 15.7 показано содержимое файла `Quake.aidl`, который нужно создать для класса Quake, реализованного выше.

Листинг 15.7. Объявление интерфейса AIDL для класса Quake

```
package com.paad.earthquake;

parcelable Quake;
```

Помните, что при передаче нестандартного объекта между процессами клиентский процесс должен иметь доступ к определению класса этого объекта.

Объявление интерфейса AIDL для Сервиса

В этом разделе вы создадите новое объявление интерфейса AIDL для Сервиса, который станет использоваться разными процессами.

Начните с образования в своем проекте нового файла с расширением `.aidl`. В нем будут храниться определения методов и полей, которые нужно включить в интерфейс, реализуемый Сервисом.

Синтаксис здесь такой же, как и у объявления обычных интерфейсов в Java.

Сперва укажите полное название пакета, затем импортируйте все пакеты, которые понадобятся. В отличие от обычных интерфейсов в Java объявление AIDL требует импорта пакетов для всех нестандартных классов и интерфейсов, даже если они входят в состав проекта.

Определите новый интерфейс, добавив свойства и методы, которые хотите сделать доступными.

Методы могут принимать несколько параметров (или не принимать их вовсе) и возвращать void или значение любого поддерживаемого типа. Если вы определили метод, принимающий один или более параметров, необходимо использовать направляющий тег в сочетании с ключевыми словами in, out и inout, чтобы задать параметр в виде значения или ссылочного типа.

ПРИМЕЧАНИЕ

По мере возможности ограничивайте направление для каждого параметра, так как процесс их передачи ресурсоемкий.

В листинге 15.8 продемонстрировано простое объявление интерфейса AIDL, которое хранится в файле `IEarthquakeService.aidl`.

Листинг 15.8. Объявление интерфейса AIDL для Сервиса EarthquakeService

```
package com.paad.earthquake;

import com.paad.earthquake.Quake;

interface IEarthquakeService {
  List<Quake> getEarthquakes();

  void refreshEarthquakes();
}
```

Реализация и расширение интерфейса межпроцессного взаимодействия

Если вы используете дополнение ATD, при сохранении файла `.aidl` автоматически сгенерируется файл, содержащий интерфейс Java. Он бу-

дет включать подкласс Stub, реализующий интерфейс в виде абстрактного класса.

Ваш Сервис должен наследовать этот класс и реализовать всю необходимую функциональность. Как правило, это делается с помощью приватного поля внутри Сервиса.

В листинге 15.9 показана реализация интерфейса IEarthquakeService, который вы объявили ранее.

Листинг 15.9. Реализация интерфейса AIDL внутри Сервиса

```
IBinder myEarthquakeServiceStub = new IEarthquakeService.Stub() {
  public void refreshEarthquakes() throws RemoteException {
    EarthquakeService.this.refreshEarthquakes();
  }

  public List<Quake> getEarthquakes() throws RemoteException {
    ArrayList<Quake> result = new ArrayList<Quake>();

    ContentResolver cr = EarthquakeService.this.getContentResolver();
    Cursor c = cr.query(EarthquakeProvider.CONTENT_URI,
                        null, null, null, null);
    if (c.moveToFirst())
      do {
        Double lat = c.getDouble(EarthquakeProvider.LATITUDE_COLUMN);
        Double lng = c.getDouble(EarthquakeProvider.LONGITUDE_COLUMN);
        Location location = new Location("dummy");
        location.setLatitude(lat);
        location.setLongitude(lng);

        String details = c.getString(EarthquakeProvider.DETAILS_COLUMN);
        String link = c.getString(EarthquakeProvider.LINK_COLUMN);

        double magnitude =
          c.getDouble(EarthquakeProvider.MAGNITUDE_COLUMN);

        long datems = c.getLong(EarthquakeProvider.DATE_COLUMN);
        Date date = new Date(datems);

        result.add(new Quake(date, details, location, magnitude, link));
      } while(c.moveToNext());
    return result;
  }
};
```

При реализации этих методов необходимо учитывать:

- что все исключения останутся локальными относительно процесса, который содержит реализацию; они не будут передаваться приложению, из которого запущен данный процесс;
- все вызовы IPC синхронные. Если вы знаете, что выполнение процесса может занять немалое время, необходимо подумать о создании

вокруг него асинхронной обертки или о перемещении ресурсоемких операций на сторону получателя, где они могут выполняться в фоновом режиме.

Реализовав необходимую функциональность, нужно открыть этот интерфейс для клиентских приложений. Для этого переопределите метод onBind внутри реализации своего Сервиса: он должен возвращать экземпляр интерфейса.

В листинге 15.10 приведена реализация метода onBind для Сервиса EarthquakeService.

Листинг 15.10. Открытие реализации интерфейса AIDL для приложений, использующих Сервис

```
@Override
public IBinder onBind(Intent intent) {
  return myEarthquakeServiceStub;
}
```

Чтобы использовать внутри Активности Сервис, обращаясь к IPC, необходимо связать его, как показано в листинге 15.11 (фрагмент кода из Активности Earthquake).

Листинг 15.11. Вызов метода из Сервиса с использованием IPC

```
IEarthquakeService earthquakeService = null;

private void bindService() {
  bindService(new Intent(IEarthquakeService.class.getName()),
              serviceConnection, Context.BIND_AUTO_CREATE);
}

private ServiceConnection serviceConnection = new ServiceConnection() {
  public void onServiceConnected(ComponentName className,
                                 IBinder service) {
    earthquakeService = IEarthquakeService.Stub.asInterface(service);
  }

  public void onServiceDisconnected(ComponentName className) {
    earthquakeService = null;
  }
};
```

Использование интернет-сервисов

Концепция программного обеспечения в виде услуги (или облачные вычисления) становится все более популярной, поскольку компании пытаются сокращать расходы на установку, обновление и поддержку разворачиваемых приложений. В результате появилось большое количество интернет-серви-

сов, для которых вы можете создавать тонкие мобильные клиенты, привнося индивидуальность с помощью мобильного телефона.

Идея использования промежуточного звена для разгрузки клиента не нова. Существует множество интернет-сервисов, предоставляющих вашим приложениям тот уровень услуг, который им требуется.

При всем желании в этой книге невозможно перечислить все разнообразие интернет-сервисов (а тем более, подробно их рассмотреть), поэтому приведем самые зрелые из них.

- **Сервисы от Google, основанные на gData.** Наряду со стандартными приложениями Google предоставляет сетевые API для доступа к таким своим Сервисам, как календарь, электронные таблицы, Blogger и Picasaweb. Все эти программные интерфейсы используют стандартный фреймворк gData от Google, представляющий собой механизм для чтения/записи данных в формате XML.
- **Yahoo! Pipes.** Сервис Yahoo! Pipes предоставляет графический инструментарий, основанный на веб-технологиях, позволяющий работать с потоками в формате XML. Используя его, вы можете фильтровать, агрегировать, анализировать и выполнять другие манипуляции с XML-потоками, преобразуя их во множество разных форматов, способных восприниматься вашими приложениями.
- **Google App Engine.** Используя Google App Engine, можете создавать облачные веб-сервисы, освобождающие ваше мобильное приложение от сложных вычислений. Это снижает нагрузку на системные ресурсы устройства, но устанавливает зависимость от стоимости подключения к Интернету.
- **Amazon Web Services.** Компания Amazon предоставляет целый ряд Сервисов, основанных на облачных вычислениях, включая мощные API для доступа к книгам, CD и DVD. Amazon также предлагает распределенное хранилище данных (S3) и Elastic Compute Cloud (EC2).

Создание насыщенных пользовательских интерфейсов

В последние годы пользовательские интерфейсы на мобильных телефонах заметно улучшились, и не в последнюю очередь благодаря iPhone с его инновационным UI.

В этом разделе вы научитесь использовать более продвинутые визуальные эффекты — шейдеры, полупрозрачность и анимацию, сочетая их с возможностями сенсорных экранов и множественными касаниями, а также применяя OpenGL. Это добавит лоска вашим Активностям и Представлениям.

Работа с анимацией

В главе 3 вы научились описывать анимацию в виде внешних ресурсов. Теперь пора использовать ее на практике.

Android предоставляет два вида анимации:

- **пошаговая** — классическая мультипликационная анимация, когда на каждый кадр отображается новый объект Drawable; рисуется внутри Представления, используя Canvas в виде проекционного экрана;
- **создаваемая путем расчета промежуточных кадров** — позволяет изменять положение, размер, поворот и прозрачность, создавая внутри Представлений эффект движущейся картинки.

ПРИМЕЧАНИЕ

Оба вида анимации ограничены изначальными рамками Представления, внутри которого они реализуются. Если повороты, преобразования и изменения масштаба приводят к тому, что содержимое не влазит в границы Представления, анимация обрезается.

Знакомство с анимацией, основанной на расчете промежуточных кадров

С помощью анимации, основанной на расчете промежуточных кадров, вы можете придать объектам на экране объем, заставить их двигаться и реагировать на действия пользователя, расходуя при этом минимальное количество ресурсов.

Используя анимацию для изменения масштаба, положения и прозрачности, вы тратите значительно меньше ресурсов, чем если бы вы выбрали ручную перерисовку объекта Canvas. Кроме того, данный метод значительно более прост в реализации.

Указанный вид анимации часто используется:

- при переходе между Активностями;
- переходе между разными разметками внутри одной Активности;
- переходе между различным содержимым, отображаемым внутри одного Представления;
- предоставлении обратной связи для пользователя:
  - отображение хода выполнения каких-то операций;
  - «сотрясание» поля ввода для оповещения о неправильных или недопустимых данных.

Создание анимации, основанной на расчете промежуточных кадров

Анимация, основанная на расчете промежуточных кадров, создается с помощью класса Animation. В приведенном списке описываются типы доступных анимационных преобразований.

- AlphaAnimation. Позволяет анимировать изменение прозрачности Представления (полупрозрачность или смешивание альфа-канала).
- RotateAnimation. Предоставляет возможность вращать выбранное Представление в плоскости XY.
- ScaleAnimation. Позволяет масштабировать Представление в обе стороны.
- TranslateAnimation. Разрешает перемещать выбранное Представление по экрану (хотя оно будет отрисовываться только в первоначальных своих границах).

Android предоставляет класс AnimationSet для группировки и настройки анимации в виде коллекции. Вы можете задать время запуска и продолжительность анимации, используемой в данной коллекции, чтобы управлять синхронизацией и порядком воспроизведения анимационной последовательности.

ПРИМЕЧАНИЕ

Важно установить начальное смещение и продолжительность для каждой дочерней анимации, иначе все они станут запускаться и завершаться одновременно.

В листингах 15.12 и 15.13 показано, как создать одну и ту же анимационную последовательность двумя способами: внутри кода программы и в виде внешнего ресурса.

Листинг 15.12. Создание анимации, основанной на расчете промежуточных кадров, программным способом

```
// Создайте объект AnimationSet
AnimationSet myAnimation = new AnimationSet(true);

// Создайте анимацию вращения.
RotateAnimation rotate = new RotateAnimation(0, 360,
  RotateAnimation.RELATIVE_TO_SELF, 0.5f,
  RotateAnimation.RELATIVE_TO_SELF, 0.5f);
rotate.setFillAfter(true);
rotate.setDuration(1000);

// Создайте анимацию масштабирования.
```

Продолжение ⇗

Листинг 15.12 (продолжение)

```
ScaleAnimation scale = new ScaleAnimation(1, 0,
                                          1, 0,
                                          ScaleAnimation.RELATIVE_TO_SELF,
                                          0.5f,
                                          ScaleAnimation.RELATIVE_TO_SELF,
                                          0.5f);
scale.setFillAfter(true);
scale.setDuration(500);
scale.setStartOffset(500);

// Создайте анимацию изменения альфа-канала.
AlphaAnimation alpha = new AlphaAnimation(1, 0);
scale.setFillAfter(true);
scale.setDuration(500);
scale.setStartOffset(500);

// Добавьте все созданные анимации в коллекцию.
myAnimation.addAnimation(rotate);
myAnimation.addAnimation(scale);
myAnimation.addAnimation(alpha);
```

В приведенном фрагменте кода реализована та же анимационная последовательность, которая показана в листинге 15.13.

Листинг 15.13. Описание анимации, основанной на расчете промежуточных кадров, с помощью XML

```
<?xml version="1.0" encoding="utf-8"?>
<set
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:shareInterpolator="true">
  <rotate
    android:fromDegrees="0" android:toDegrees="360"
    android:pivotX="50%" android:pivotY="50%"
    android:startOffset="0"
    android:duration="1000" />
  <scale
    android:fromXScale="1.0"
    android:toXScale="0.0"
    android:fromYScale="1.0"
    android:toYScale="0.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="500"
    android:duration="500" />
  <alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:startOffset="500"
    android:duration="500" />
</set>
```

Как видно, создание анимационных последовательностей становится более простым и интуитивным, если использовать внешние ресурсы.

Применение анимации, основанной на расчете промежуточных кадров

Анимация может использоваться в сочетании с любым Представлением. Делается это с помощью метода startAnimation, которому в качестве параметров передаются объекты Animation или AnimationSet.

Анимационные последовательности остановятся после первого воспроизведения, однако такое поведение можно изменить, используя методы setRepeatMode и setRepeatCount, вызываемые из тех же объектов Animation и AnimationSet. Вы можете зациклить анимацию или заставить ее повторяться в обратном порядке, используя режимы RESTART и REVERSE соответственно. Параметр для метода setRepeatCount будет указывать, сколько раз анимация должна повториться.

В листинге 15.14 показана анимация, повторяющаяся бесконечно.

Листинг 15.14. Применение непрерывно повторяющейся анимации

```
myAnimation.setRepeatMode(Animation.RESTART);
myAnimation.setRepeatCount(Animation.INFINITE);
myView.startAnimation(myAnimation);
```

Использование AnimationListener

Интерфейс AnimationListener позволяет создать обработчик событий, который срабатывает в начале или при завершении анимации. Используя его, вы можете совершать какие-либо операции, прежде чем (или после того как) анимация закончит работу. Это может быть изменение содержимого Представления или последовательный показ нескольких анимаций.

Вызовите метод setAnimationListener из объекта Animation и передайте ему в качестве параметра реализацию интерфейса AnimationListener, при необходимости переопределив методы onAnimationEnd, onAnimationStart и onAnimationRepeat.

В листинге 15.15 наглядно продемонстрирована простая реализация интерфейса AnimationListener.

Листинг 15.15. Создание реализации AnimationListener

```
myAnimation.setAnimationListener(new AnimationListener() {
  public void onAnimationEnd(Animation _animation) {
    // TODO Какие-то действия после завершения анимации.
  }

  public void onAnimationRepeat(Animation _animation) {
    // TODO Какие-то действия при повторении анимации.
  }

  public void onAnimationStart(Animation _animation) {
    // TODO Какие-то действия при запуске анимации.
  }
});
```

Пример изменения пользовательского интерфейса с помощью анимации

В этом примере[1] вы создадите новую Активность, в которой анимация используется для плавного изменения содержимого пользовательского интерфейса в зависимости от направления, куда повернут манипулятор D-pad.

1. Начните с создания нового проекта ContentSlider и одноименной Активности для него.

```
package com.paad.contentslider;

import android.app.Activity;
import android.view.KeyEvent;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.Animation.AnimationListener;
import android.view.animation.AnimationUtils;
import android.widget.TextView;

public class ContentSlider extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
}
```

2. Отредактируйте ресурс с разметкой `main.xml`. Он должен содержать один элемент TextView с жирным центрированным текстом относительно большого размера.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:id="@+id/myTextView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:textStyle="bold"
    android:textSize="30sp"
    android:text="CENTER"
    android:editable="false"
    android:singleLine="true"
    android:layout_margin="10dp"
  />
```

[1] Все фрагменты кода в этом примере — часть проекта Animated Slider из данной главы, их можно загрузить с сайта Wrox.com.

```
</LinearLayout>
```

3. Создайте анимационную последовательность, при которой текущее Представление «выезжает» за пределы экрана и на его место «въезжает» другое. Задайте рамки для каждого направления: лево, право, верх и низ. Каждая анимация должна содержаться в отдельном файле.

   3.1. Создайте файл `slide_bottom_in.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <translate
    android:fromYDelta="-100%p"
    android:toYDelta="0"
    android:duration="700"
  />
</set>
```

   3.2. Создайте файл `slide_bottom_out.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <translate
    android:fromYDelta="0"
    android:toYDelta="100%p"
    android:duration="700"
  />
</set>
```

   3.3. Создайте файл `slide_top_in.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <translate
    android:fromYDelta="100%p"
    android:toYDelta="0"
    android:duration="700"
  />
</set>
```

   3.4. Создайте файл `slide_top_out.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <translate
    android:fromYDelta="0"
    android:toYDelta="-100%p"
    android:duration="700"
  />
</set>
```

   3.5. Создайте файл `slide_left_in.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
```

```
  <translate
    android:fromXDelta="100%p"
    android:toXDelta="0"
    android:duration="700"
  />
</set>
```

3.6. Создайте файл `slide_left_out.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <translate
    android:fromXDelta="0"
    android:toXDelta="-100%p"
    android:duration="700"
  />
</set>
```

3.7. Создайте файл `slide_right_in.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <translate
    android:fromXDelta="-100%p"
    android:toXDelta="0"
    android:duration="700"
  />
</set>
```

3.8. Создайте файл `slide_right_out.xml`.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <translate
    android:fromXDelta="0"
    android:toXDelta="100%p"
    android:duration="700"
  />
</set>
```

4. Вернитесь к Активности ContentSlider и извлеките ссылки на элемент TextView и на все ресурсы с анимацией, которые вы создали в пункте 3.

```
Animation slideInLeft;
Animation slideOutLeft; Animation slideInRight;
Animation slideOutRight;
Animation slideInTop;
Animation slideOutTop;
Animation slideInBottom;
Animation slideOutBottom;
TextView myTextView;

@Override
public void onCreate(Bundle savedInstanceState) {
```

```
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  slideInLeft = AnimationUtils.loadAnimation(this,
    R.anim.slide_left_in);
  slideOutLeft = AnimationUtils.loadAnimation(this,
    R.anim.slide_left_out);
  slideInRight = AnimationUtils.loadAnimation(this,
    R.anim.slide_right_in);
  slideOutRight = AnimationUtils.loadAnimation(this,
    R.anim.slide_right_out);
  slideInTop = AnimationUtils.loadAnimation(this,
    R.anim.slide_top_in);
  slideOutTop = AnimationUtils.loadAnimation(this,
    R.anim.slide_top_out);
  slideInBottom = AnimationUtils.loadAnimation(this,
    R.anim.slide_bottom_in);
  slideOutBottom = AnimationUtils.loadAnimation(this,
    R.anim.slide_bottom_out);

  myTextView = (TextView)findViewById(R.id.myTextView);
}
```

Каждый экранный переход состоит из двух связанных анимаций: перемещения старого текста за пределы экрана и появления нового на его месте. Вместо того чтобы создавать несколько Представлений, можно изменить свойства уже имеющегося, когда оно выходит за пределы экрана, и поместить его обратно уже с другой стороны.

5. Создайте новый метод, который применяет анимацию скрытия и ждет ее завершения, прежде чем изменить текст и инициировать анимацию появления.

```
private void applyAnimation(Animation _out,
                            Animation _in,
                            String _newText) {
  final String text = _newText;
  final Animation in = _in;

  // Убедитесь, что текст остается за рамками экрана,
  // когда анимация скрытия завершена.
  _out.setFillAfter(true);

  // С помощью AnimationListener ожидайте завершение
  // анимации скрытия.
  _out.setAnimationListener(new AnimationListener() {
    public void onAnimationEnd(Animation _animation) {
      // Измените текст
      myTextView.setText(text);
      // Верните его обратно на экран
      myTextView.startAnimation(in);
    }

    public void onAnimationRepeat(Animation _animation) {}
```

```
    public void onAnimationStart(Animation _animation) {}
  });

  // Примените анимацию скрытия
  myTextView.startAnimation(_out);
}
```

6. Отображаемый текст может иметь девять позиций. Создайте перечисление (enum) для каждой из них, а также отдельное поле, чтобы отслеживать текущее положение Представления.

```
TextPosition textPosition = TextPosition.Center;
enum TextPosition { UpperLeft, Top, UpperRight,
                    Left, Center, Right,
                    LowerLeft, Bottom, LowerRight };
```

7. Создайте новый метод movePosition, который в качестве параметров принимает текущую позицию и направление движения, вычисляя новое положение на экране. В конце он должен запускать анимационную последовательность, созданную в пункте 5.

```
private void movePosition(TextPosition _current,
                          TextPosition _directionPressed) {
  Animation in;
  Animation out;
  TextPosition newPosition;

  if (_directionPressed == TextPosition.Left){
    in = slideInLeft;
    out = slideOutLeft;
  }
  else if (_directionPressed == TextPosition.Right){
    in = slideInRight;
    out = slideOutRight;
  }
  else if (_directionPressed == TextPosition.Top){
    in = slideInTop;
    out = slideOutTop;
  }
  else {
    in = slideInBottom;
    out = slideOutBottom;
  }

  int newPosValue = _current.ordinal();
  int currentValue = _current.ordinal();

  // Чтобы имитировать эффект "наклона" устройства, движение в заданном
  // направлении должно приводить к появлению текста из противоположной
  // стороны. То есть при наклоне вправо текст должен появляться слева.
  if (_directionPressed == TextPosition.Bottom)
    newPosValue = currentValue - 3;
  else if (_directionPressed == TextPosition.Top)
    newPosValue = currentValue + 3;
  else if (_directionPressed == TextPosition.Right) {
    if (currentValue % 3 != 0)
```

```
        newPosValue = currentValue - 1;
    }
    else if (_directionPressed == TextPosition.Left) {
      if ((currentValue+1) % 3 != 0)
        newPosValue = currentValue + 1;
    }

    if (newPosValue != currentValue &&
        newPosValue > -1 &&
        newPosValue < 9){
        newPosition = TextPosition.values()[newPosValue];

      applyAnimation(in, out, newPosition.toString());
      textPosition = newPosition;
    }
  }
```

8. Подключите манипулятор D-pad. Для этого нужно переопределить обработчик onKeyDown из Активности, чтобы отслеживать нажатия и в итоге вызывать метод movePosition.

```
@Override
public boolean onKeyDown(int _keyCode, KeyEvent _event) {
  if (super.onKeyDown(_keyCode, _event))
    return true;

  if (_event.getAction() == KeyEvent.ACTION_DOWN){
    switch (_keyCode) {
      case (KeyEvent.KEYCODE_DPAD_LEFT):
        movePosition(textPosition, TextPosition.Left); return true;
      case (KeyEvent.KEYCODE_DPAD_RIGHT):
        movePosition(textPosition, TextPosition.Right); return true;
      case (KeyEvent.KEYCODE_DPAD_UP):
        movePosition(textPosition, TextPosition.Top); return true;
      case (KeyEvent.KEYCODE_DPAD_DOWN):
        movePosition(textPosition, TextPosition.Bottom);
        return true;
    }
  }
  return false;
}
```

После запуска приложение отобразит центрированный текст. Поворот манипулятора в одном из четырех направлений переместит текст в соответствующую позицию.

ПРИМЕЧАНИЕ

В качестве дополнения можете подключить акселерометр вместо использования манипулятора D-pad.

Анимация разметки и Групп представлений

Класс LayoutAnimation нужен для применения анимации к Группам представлений: передает одиночный объект Animation (или AnimationSet)

для каждого дочернего Представления в заранее определенной последовательности.

Используйте объект LayoutAnimationController для задания объекта Animation (или AnimationSet), который применяется к каждому Представлению в группе. Ко всем Представлениям, содержащим этот объект, будет применена одна и та же анимация, но вы можете задействовать LayoutAnimationController для указания порядка и времени запуска для каждого Представления.

Для этих целей в Android предусмотрено два класса:

- LayoutAnimationController — позволяет выбирать начальный сдвиг для каждого Представления (в миллисекундах) и порядок (нормальный, обратный или случайный), в каком анимация применится для каждого дочернего элемента;
- GridLayoutAnimationController — как производный класс назначает анимационную последовательность для дочерних Представлений с использованием ссылки на строки и столбцы.

Создание анимации для разметки. Чтобы создать новую анимацию для разметки, нужно определить ее для каждого дочернего Представления. После создайте новый элемент LayoutAnimation (либо в коде программы, либо в виде внешнего ресурса), ссылающийся на ранее выполненную анимацию и описывающий порядок и время срабатывания, с которыми ее необходимо применить.

В листинге 15.16 показано определение простой анимации, она хранится в виде файла `popin.xml` в каталоге res/anim, а также описание LayoutAnimation из файла `popinlayout.xml`.

Элемент LayoutAnimation в случайном порядке применяет всплывающую анимацию к каждому дочернему Представлению заданной группы.

Листинг 15.16. Создание анимации для разметки

res/anim/popin.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
  <scale
    android:fromXScale="0.0" android:toXScale="1.0"
    android:fromYScale="0.0" android:toYScale="1.0"
    android:pivotX="50%" android:pivotY="50%"
    android:duration="400"
  />
</set>
```

res/anim/popinlayout.xml

```
<layoutAnimation
  xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:delay="0.5"
    android:animationOrder="random"
    android:animation="@anim/popin"
  />
```

Использование анимации для разметки. После описания элемента LayoutAnimation можете применить его к Группе представлений (в коде программы или с помощью ресурса в формате XML). В XML это делается путем добавления атрибута `android:layoutAnimation` к определению разметки:

```
android:layoutAnimation="@anim/popinlayout"
```

Чтобы применить анимацию для разметки в коде программы, вызовите метод setLayoutAnimation из объекта ViewGroup, передавая ему ссылку на соответствующий экземпляр LayoutAnimation.

В обоих случаях анимация выполнится один раз, при первом появлении Группы представлений. Вы можете вызвать ее повторно, запустив метод scheduleLayoutAnimation из объекта ViewGroup. Таким образом, анимация выполнится при следующем появлении Группы представлений.

Анимация для разметки также имеет поддержку интерфейса AnimationListener.

В листинге 15.17 показан повторный запуск анимации в контексте Группы представлений с помощью интерфейса AnimationListener, который используется для выполнения дополнительных действий по завершении изначальной анимации.

Листинг 15.17. Использование интерфейса AnimationListener и анимации для разметки

```
aViewGroup.setLayoutAnimationListener(new AnimationListener() {
  public void onAnimationEnd(Animation _animation) {
    // TODO: Действия, выполняемые по завершении выполнения анимации.
  }
  public void onAnimationRepeat(Animation _animation) {}
  public void onAnimationStart(Animation _animation) {}
});

aViewGroup.scheduleLayoutAnimation();
```

Создание и использование пошаговой анимации

Пошаговая анимация напоминает классические мультипликационные фильмы, где каждому кадру соответствовало отдельное изображение. Если анимация, основанная на расчете промежуточных кадров, использует определенное Представление для отображения содержимого, то пошаговая версия позволяет задать последовательность объектов Drawable, которые станут фоном для Представления.

Класс AnimationDrawable нужен для новой пошаговой анимации, представленной в виде ресурса Drawable. Вы можете задать его в виде внешнего XML-ресурса в каталоге res/drawable своего проекта.

Используйте тег `<animation-list>` для группировки набора узлов `<item>`, каждый из которых применяет атрибут `drawable` для задания отображаемой картинки, а также duration, чтобы указать время, на протяжении которого она будет показываться.

В листинге 15.18 демонстрируется процесс создания простой анимации, которая отображает взлет ракеты (изображения ракеты не включены). Соответствующий файл сохранен под именем `animated_rocket.xml` в каталоге res/drawable.

Листинг 15.18. Создание пошаговой анимации с помощью XML

```
<animation-list
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:oneshot="false">
  <item android:drawable="@drawable/rocket1" android:duration="500" />
  <item android:drawable="@drawable/rocket2" android:duration="500" />
  <item android:drawable="@drawable/rocket3" android:duration="500" />
</animation-list>
```

Чтобы отобразить вашу анимацию, задайте ее в виде фона для Представления, применив метод setBackgroundResource:

```
ImageView image = (ImageView)findViewById(R.id.my_animation_frame);
image.setBackgroundResource(R.drawable.animated_rocket);
```

Вы можете также использовать метод setBackgroundDrawable, чтобы указывать экземпляр Drawable вместо ссылки на ресурс. Запустите анимацию, вызвав метод start:

```
AnimationDrawable animation = (AnimationDrawable)image.getBackground();
animation.start();
```

Продвинутое рисование с помощью Canvas

С классом Canvas вы познакомились в главе 4, когда учились создавать собственные Представления. Canvas также упоминался в главе 8 при размещении аннотаций на Наложениях для элемента MapView.

Концепция Холста — распространенная абстракция, используемая в программировании графики и, как правило, состоящая из трех компонентов:

- Canvas — предоставляет методы для рисования, которые отображают графические примитивы на исходном растровом изображении;

- Paint — иногда встречается под названием brush («кисть»); Paint позволяет указывать, как именно графические примитивы должны отображаться на растровом изображении;
- Bitmap — поверхность, на которой происходит рисование.

Большинство продвинутых методик, описанных в этой главе, связаны с изменением и модификацией объекта Paint, с помощью которого вы можете придавать объем и текстуру плоским растровым изображениям.

API для рисования в Android поддерживает полупрозрачность, градиентные заливки, округленные прямоугольники и сглаживание. К сожалению, из-за ограниченных ресурсов векторная графика пока что не поддерживается, вместо этого используется традиционная растровая перерисовка.

В результате такого подхода возрастает эффективность, изменение объекта Paint влияет только на новые элементы, не затрагивая нарисованные.

ПРИМЕЧАНИЕ

Для тех, у кого есть опыт разработки под Windows: возможности рисования двумерной (2D) графики в Android мало чем отличаются от тех, что предоставляет библиотека GDI+.

Что вы можете нарисовать

Класс Canvas можно назвать оберткой вокруг растрового изображения, которое вы будете использовать в качестве полотна для своих художественных изысков. Он предоставляет набор методов вида draw* для воплощения задумок.

В приведенном списке кратко указаны доступные графические примитивы.

- drawARGB/drawRGB/drawColor. Заполняет Холст сплошным цветом.
- drawArc. Рисует дугу между двумя углами внутри заданной прямоугольной области.
- drawBitmap. Рисует растровое изображение на элементе Canvas. Вы можете изменять внешний вид целевой картинки, указывая итоговый размер или используя матрицу для преобразования.
- drawBitmapMesh. Рисует изображение с использованием сетки, с помощью которой можно управлять отображением итоговой картинки, перемещая точки внутри нее.
- drawCircle. Рисует окружность с определенным радиусом вокруг заданной точки.

- drawLine(s). Рисует линию (или последовательность линий) между двумя точками.
- drawOval. Рисует овал на основе прямоугольной области.
- drawPaint. Закрашивает весь Холст с помощью заданного объекта Paint.
- drawPath. Рисует указанный контур, используется для хранения набора графических примитивов в виде единого объекта.
- drawPicture. Рисует объект Picture внутри заданного прямоугольника.
- drawPosText. Рисует текстовую строку, учитывая смещение для каждого символа.
- drawRect. Рисует прямоугольник.
- drawRoundRect. Рисует прямоугольник с закругленными углами.
- drawText. Рисует текстовую строку на Холсте. Шрифт, размер, цвет и свойства отображения текста задаются в соответствующем объекте Paint.
- drawTextOnPath. Рисует текст, который отображается вокруг определенного контура.
- drawVertices. Рисует набор треугольников в виде совокупности вершинных (вертексных) точек.

Каждый из этих методов позволяет указать объект Paint для отображения нарисованного. В следующих разделах вы научитесь создавать и изменять объекты Paint, чтобы получить максимальную отдачу от своих зарисовок.

Извлечение максимальной пользы из объекта Paint

Класс Paint представляет собой сочетание кисти и палитры. Он позволяет выбирать способ отображения графических примитивов, которые вы рисуете на объекте Canvas с помощью методов, описанных в предыдущем разделе. Изменяя объект Paint, можно контролировать цвет, стиль, шрифт и специальные эффекты, используемые при рисовании.

Метод setColor позволяет выбрать цвет кисти, стиль объекта Paint (задаваемый с помощью метода setStyle) — рисовать либо очертания графического примитива (STROKE), либо его заливку (FILL), либо и то, и другое сразу (STROKE_AND_FILL).

Помимо этих простых методов класс Paint поддерживает прозрачность и может быть изменен с помощью различных шейдеров, фильтров и эффектов, предоставляет богатый набор сложных красок и кистей.

Android SDK включает в себя проекты, предоставляющие большую часть возможностей класса Paint. Они доступны в подкаталоге graphics наряду с другими приложениями, демонстрирующими различные API:

```
[корневой каталог sdk]\samples\ApiDemos\src\com\android\samples\graphics
```

В следующих разделах вы узнаете, как пользоваться некоторыми из этих возможностей, и испробуете их на практике (например, научитесь закруглять углы и работать с градиентами), не углубляясь во все возможные комбинации.

Использование полупрозрачности. Любой цвет в Android содержит свойство прозрачности (альфа-канал).

Указать его можно при создании описывающей цвет переменной, используя методы argb и parseColor:

```
// Сделайте цвет красным и наполовину прозрачным
int opacity = 127;
int intColor = Color.argb(opacity, 255, 0, 0);
int parsedColor = Color.parseColor("#7FFF0000");
```

Как вариант, можете задать прозрачность уже существующего объекта Paint с помощью метода setAlpha:

```
// Сделайте цвет наполовину прозрачным
int opacity = 127;
myPaint.setAlpha(opacity);
```

Создание цвета, непрозрачность которого менее 100 %, означает, что любой графический примитив, нарисованный с его помощью, станет частично прозрачным: в какой-то мере будет видно все, что нарисовано под ним.

Вы можете использовать эффект прозрачности в любом классе или методе, которые работают с цветом, включая Paint, Shader, MaskFilter.

Знакомство с шейдерами. Расширения класса Shader позволяют создавать объекты Paint, которые закрашивают элементы более сложным образом, чем просто заливка сплошным цветом.

Наиболее часто шейдеры используются для описания градиентной заливки. При помощи градиентов вы можете легко придать двумерному рисунку глубину и фактуру. Android поддерживает три градиентных шейдера (помимо растровых и композитных).

Описывать методики рисования словами — занятие бессмысленное по своей сути, поэтому лучше взгляните на рис. 15.1, чтобы понять принцип работы каждого из этих шейдеров. Слева направо представлен результат работы классов LinearGradient, RadialGradient и SweepGradient.

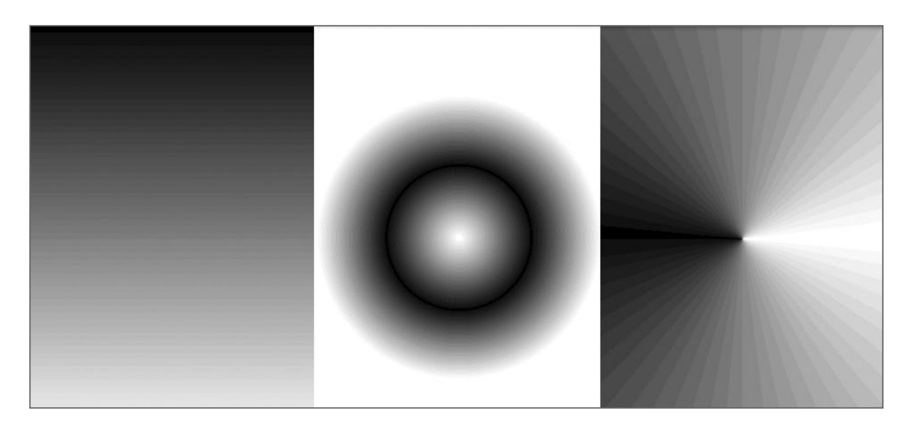


Рис. 15.1.

ПРИМЕЧАНИЕ

На рис. 15.1 не представлены классы ComposeShader и BitmapShader. С помощью первого вы можете смешивать разные шейдеры, второй позволяет создавать кисти для рисования, основанные на растровых изображениях.

Чтобы использовать шейдеры при рисовании, их нужно применять к объекту Paint с помощью метода setShader:

```
Paint shaderPaint = new Paint();
shaderPaint.setShader(myLinearGradient);
```

Все, что вы нарисуете с применением этой кисти, будет закрашиваться указанным шейдером, вместо сплошного цвета.

Задание градиентных шейдеров. Как говорилось в предыдущих разделах, градиентные шейдеры позволяют закрашивать контуры с помощью интерполированной цветовой гаммы. Вы можете задать градиент двумя способами.

Первый — простой переход между двумя цветами, как показано в листинге 15.19 на примере класса LinearGradientShader.

Листинг 15.19. Создание линейного градиента

```
int colorFrom = Color.BLACK;
int colorTo = Color.WHITE;

LinearGradient linearGradientShader = new LinearGradient(x1, y1, x2, y2,
                                                          colorFrom,
                                                          colorTo,
                                                          TileMode.CLAMP);
```

Второй способ, продемонстрированный в листинге 15.20, предусматривает описание более сложных последовательностей цветов, распределенных в заданных пропорциях.

Листинг 15.20. Создание радиального градиента

```
int[] gradientColors = new int[3];
gradientColors[0] = Color.GREEN;
gradientColors[1] = Color.YELLOW;
gradientColors[2] = Color.RED;

float[] gradientPositions = new float[3];
gradientPositions[0] = 0.0f;
gradientPositions[1] = 0.5f; gradientPositions[2] = 1.0f;

RadialGradient radialGradientShader = new RadialGradient(centerX, centerY,
                                                         radius,
                                                         gradientColors,
                                                         gradientPositions,
                                                         TileMode.CLAMP);
```

Каждый из градиентных шейдеров (линейный, радиальный и развернутый) позволяет задать градиентную заливку, используя любой из приведенных выше способов.

Использование режимов заполнения в шейдерах. Размеры кисти для градиентных шейдеров определяются с помощью явно заданных прямоугольных границ или с использованием сочетания центральной точки и длины радиуса. В растровых шейдерах размер кисти равен размерам изображения.

Если область, заданная кистью шейдера, меньше, чем область для заполнения, режим TileMode определяет, каким образом заполнится остальная площадь.

- CLAMP. Использует цвета по краям шейдера для заполнения дополнительного пространства.
- MIRROR. Отображает изображение шейдера по горизонтали и вертикали таким образом, чтобы каждая следующая часть соприкасалась с предыдущей.
- REPEAT. Повторяет изображение шейдера по горизонтали и вертикали, но не отображает его.

Использование фильтров для масок. Класс MaskFilter позволяет назначать контурные эффекты для объекта Paint. Классы, наследующие MaskFilter, применяют преобразования к альфа-каналу объекта Paint вдоль его внешней границы.

Android включает следующие фильтры для масок:

- BlurMaskFilter — задает стиль размытия и радиус выступов для краев объекта Paint;
- EmbossMaskFilter — задает направление источника света и уровень освещенности, создавая эффект рельефа.

Чтобы применить фильтр для маски, используйте метод setMaskFilter, передавая ему в качестве параметра объект MaskFilter. В листинге 15.21 показан процесс наложения фильтра EmbossMaskFilter на имеющийся объект Paint.

Листинг 15.21. Применение фильтра EmbossMaskFilter к объекту Paint

```
// Задайте направление источника света
float[] direction = new float[]{ 1, 1, 1 };
// Установите уровень освещенности
float light = 0.4f;
// Выберите степень зеркальности
float specular = 6;
// Укажите, какой уровень размытия должен применяться к маске
float blur = 3.5f;
EmbossMaskFilter emboss = new EmbossMaskFilter(direction, light,
                                               specular, blur);

// Примените маску
myPaint.setMaskFilter(emboss);
```

Демонстрационный проект FingerPaint, входящий в состав SDK, — отличный пример использования объектов MaskFilter. Он демонстрирует оба эффекта — BlurMaskFilter и EmbossMaskFilter.

Использование цветовых фильтров. В отличие от фильтров для масок, которые преобразуют альфа-канал объекта Paint, цветовые фильтры затрагивают каждый из каналов RGB. Все потомки класса ColorFilter игнорируют альфа-канал во время преобразований.

Android содержит три цветовых фильтра.

- ColorMatrixColorFilter. Позволяет задать для объекта Paint матрицу ColorMatrix размером 4 x 5. Объекты ColorMatrix, как правило, используются при программной обработке изображений, могут пригодиться также для последовательных преобразований с применением умножения матриц.
- LightingColorFilter. Умножает каналы RGB первого цвета, прежде чем добавить второй. Результат каждого преобразования варьируется от 0 до 255.
- PorterDuffColorFilter. Предлагает воспользоваться одним из шестнадцати режимов смешивания цифровых изображений Портера-Даффа, чтобы применить заданный цвет к объекту Paint.

Использовать цветовые фильтры можно, задействовав для этого метод setColorFilter:

```
myPaint.setColorFilter(new LightingColorFilter(Color.BLUE, Color.RED));
```

В каталоге с примерами вы найдете приложение под названием ColorMatrixSample, которое демонстрирует работу цветовых фильтров и матриц.

Использование контурных эффектов. До сих пор рассматривались эффекты, которые влияли на способ заливки рисунка. Контурные эффекты используются для управления отрисовкой контура. Они чрезвычайно полезны для рисования контурных графических примитивов, но могут быть применены к любому объекту Paint, чтобы повлиять на способ отрисовки их очертаний.

Используя этот вид эффектов, вы можете менять внешний вид углов фигур и их очертание. Android содержит несколько контурных эффектов.

- CornerPathEffect. Позволяет сглаживать острые углы в форме графического примитива, заменяя их на закругленные.
- DashPathEffect. Вместо рисования сплошного контура можете использовать DashPathEffect для создания очертания, состоящего из ломаных линий (тире/точек). Есть возможность указать любой шаблон повторения сплошных/пустых отрезков.
- DiscretePathEffect. Делает то же самое, что и DashPathEffect, но добавляет элемент случайности. Указываются длина каждого отрезка и степень отклонения от оригинального контура.
- PathDashPathEffect. Позволяет определить новую фигуру (контур), чтобы использовать ее в виде отпечатка оригинального контура.

Следующие классы помогают сочетать различные контурные эффекты в контексте единственного объекта Paint:

- SumPathEffect — добавляет последовательность из двух эффектов, каждый из которых применяется к оригинальному контуру, после чего результаты смешиваются;
- ComposePathEffect — использует первый эффект, затем к полученному результату добавляет второй.

Контурные эффекты, влияющие на форму объекта, который должен быть нарисован, изменяют и область, занимаемую им. Благодаря этому любые эффекты для закрашивания, применяемые к данной фигуре, отрисовываются в новых границах.

Контурные эффекты применяются к объекту Paint с помощью метода setPathEffect:

```
borderPaint.setPathEffect(new CornerPathEffect(5));
```

В каталоге с примерами вы можете найти руководство для работы с каждым из описанных выше эффектов.

Изменение режима Xfermode. Изменение режима Xfermode для объекта Paint влияет на способ наложения новых цветов поверх уже нарисованных.

В обычных обстоятельствах при рисовании поверх имеющегося рисунка создастся новый верхний слой. Если новый объект Paint на 100 % непрозрачный, он полностью закрасит все, что находится под областью для рисования; если он полупрозрачный, то только затенит лежащие ниже цвета.

Подклассы Xfermode позволяют изменить такое поведение.

- AvoidXfermode. Определяет цвет, поверх которого объект Paint не может (или наоборот — может только поверх него) рисовать. Задается также параметр tolerance, указывающий на допустимое отклонение.
- PixelXorXfermode. Применяет простое побитовое исключение (XOR) при рисовании поверх существующих цветов.
- PorterDuffXfermode. Мощный режим, с помощью которого можно использовать любое из шестнадцати правил смешивания изображений Портера-Даффа, управляя процессом наложения кисти на уже существующий рисунок.

Для того чтобы применить один из этих режимов, используйте метод setXferMode:

```
AvoidXfermode avoid = new AvoidXfermode(Color.BLUE, 10,
                                         AvoidXfermode.Mode.AVOID);
borderPen.setXfermode(avoid);
```

Улучшение качества отображения с помощью сглаживания

При создании нового объекта Paint вы можете передать в его конструктор несколько флагов, которые будут влиять на способ отображения. Одним из наиболее интересных из них считается флаг ANTI_ALIAS_FLAG, обеспечивающий сглаживание диагональных линий, рисуемых объектом Paint (снижая при этом производительность).

Сглаживание играет важную роль в процессе отрисовки текста, значительно упрощает его восприятие. Чтобы сделать текст еще более гладким, можете использовать флаг SUBPIXEL_TEXT_FLAG, который применяет субпиксельное сглаживание.

Можно задать оба этих флага вручную, используя методы setSubpixelText и setAntiAlias:

```
myPaint.setSubpixelText(true);
myPaint.setAntiAlias(true);
```

Рекомендации по использованию объекта Canvas для рисования

Двумерные операции для рисования, как правило, расходуют много ценных процессорных ресурсов; неэффективное их использование может заблокировать графический поток и негативно сказаться на отзывчивости приложения. Это особенно актуально для систем с ограниченными ресурсами, с одним слабым процессором.

Нужно следить за расходом ресурсов и ценить каждый такт центрального процессора, потребляемый вашим методом onDraw, чтобы не получить в итоге абсолютно неотзывчивое приложение.

Существует множество методик, которые помогают минимизировать потребление ресурсов, связанное с процессом отрисовки. Далее приводится несколько советов, с помощью которых вы сможете создавать хорошо выглядящие и отзывчивые Активности (этот список неисчерпывающий).

- **Учитывайте размер и ориентацию.** Разрабатывая Представления и Наложения, не забывайте учитывать и проверять то, как они выглядят при разных разрешениях, размерах экрана и плотностях пикселов.
- **Создавайте статические объекты только один раз.** Создание объектов и сбор мусора — чрезвычайно «дорогие» операции. По возможности создавайте объекты для рисования, такие как Paint, Path и Shader, всего один раз, избегая такой повторой операции при каждой перерисовке Представления.
- **Помните, что метод onDraw потребляет много ресурсов.** Выполнение метода onDraw — ресурсоемкий процесс, в ходе которого операционная система вынуждена проводить какое-то количество операций по совмещению и созданию растровых изображений. Ниже представлено несколько способов, с помощью которых можно изменять внешний вид объекта Canvas без перерисовки.
  - **Проведите преобразования Холста.** Используйте такие методы, как rotate и translate для упрощения сложного взаимного позиционирования элементов на вашем Холсте. К примеру, вместо позиционирования и вращения текстового элемента вокруг циферблата часов поверните Холст на 22,5° и нарисуйте текст на том же месте.
  - **Используйте анимацию.** Она понадобится для проведения заранее заданных преобразований вашего Представления, вместо того чтобы перерисовывать его вручную. Операции масштабирования, поворота и перемещения можно выполнить для любого Представления внутри Активности. В итоге ресурсы расходуются эффективно.
  - **Применяйте растровые изображения, картинки в формате 9-patch и ресурсы Drawable.** Если у ваших Представлений статический

фон, стоит обратить внимание на растровые, масштабируемые (NinePatch) и статические изображения, заданные в формате XML, вместо того чтобы создавать их динамически.

Пример усовершенствованного циферблата для компаса

В главе 4 вы создали простой пользовательский интерфейс для компаса. В главе 14 вы вернулись к нему, добавив отображение параметров pitch и roll, используя аппаратные возможности акселерометра.

Для этих примеров был создан довольно примитивный пользовательский интерфейс, чтобы не засорять код лишними деталями.

В следующем примере[1] вы внесете некоторые существенные изменения в метод onDraw из Представления CompassView, превращая простой, плоский компас в динамический автогоризонт, показанный на рис. 15.2.

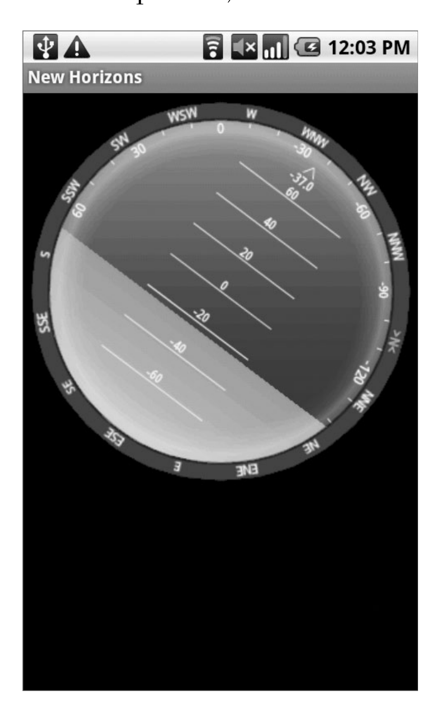


Рис. 15.2.

Так как рисунки в этой книге черно-белые, чтобы увидеть новый компас во всей красе, вам самим придется его создать.

1. Начните с редактирования файла `colors.xml`, добавляя значения цветов для градиента окантовки, затенения стекла компаса, неба

[1] Все фрагменты кода в этом примере — часть проекта по автогоризонту из данной главы, их можно загрузить с сайта Wrox.com.

и земли. Измените цвета, используемые для обозначений на окантовке и циферблате.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="text_color">#FFFF</color>
  <color name="background_color">#F000</color>
  <color name="marker_color">#FFFF</color>
  <color name="shadow_color">#7AAA</color>
  <color name="outer_border">#FF444444</color>
  <color name="inner_border_one">#FF323232</color>
  <color name="inner_border_two">#FF414141</color>
  <color name="inner_border">#FFFFFFFF</color>
  <color name="horizon_sky_from">#FFA52A2A</color>
  <color name="horizon_sky_to">#FFFFC125</color>
  <color name="horizon_ground_from">#FF5F9EA0</color>
  <color name="horizon_ground_to">#FF00008B</color>
</resources>
```

2. Объекты Paint и Shader, использованные для отображения неба и земли в автогоризонте, создаются на основе размеров текущего Представления, поэтому они нестатические (в отличие от объектов Paint, которые вы создали в главе 4). Вместо работы с объектом Paint определите массив градиентов, а также несколько цветов для них.

```
int[] borderGradientColors;
float[] borderGradientPositions;

int[] glassGradientColors;
float[] glassGradientPositions;

int skyHorizonColorFrom;
int skyHorizonColorTo;
int groundHorizonColorFrom;
int groundHorizonColorTo;
```

3. Примените метод initCompassView из CompassView для инициализации полей, созданных вами в предыдущем пункте. Воспользуйтесь для этого ресурсами из пункта 1. Уже написанный код можно оставить почти нетронутым, внеся лишь некоторые изменения в переменные textPaint, circlePaint и markerPaint, как отмечено в следующем фрагменте.

```
protected void initCompassView() {
  setFocusable(true);
  // Получите внешние ресурсы
  Resources r = this.getResources();

  circlePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  circlePaint.setColor(R.color.background_color);
  circlePaint.setStrokeWidth(1);
  circlePaint.setStyle(Paint.Style.STROKE);

  northString = r.getString(R.string.cardinal_north);
```

```
eastString = r.getString(R.string.cardinal_east);
southString = r.getString(R.string.cardinal_south);
westString = r.getString(R.string.cardinal_west);

textPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
textPaint.setColor(r.getColor(R.color.text_color));
textPaint.setFakeBoldText(true);
textPaint.setSubpixelText(true);
textPaint.setTextAlign(Align.LEFT);

textHeight = (int)textPaint.measureText("yY");

markerPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
markerPaint.setColor(r.getColor(R.color.marker_color));
markerPaint.setAlpha(200);
markerPaint.setStrokeWidth(1);
markerPaint.setStyle(Paint.Style.STROKE);
markerPaint.setShadowLayer(2, 1, 1, r.getColor(R.color.shadow_color));
```

3.1. Создайте массивы для хранения цветов и позиций, которые будут использованы радиальным шейдером для рисования окантовки.

```
borderGradientColors = new int[4];
borderGradientPositions = new float[4];

borderGradientColors[3] = r.getColor(R.color.outer_border);
borderGradientColors[2] = r.getColor(R.color.inner_border_one);
borderGradientColors[1] = r.getColor(R.color.inner_border_two);
borderGradientColors[0] = r.getColor(R.color.inner_border);
borderGradientPositions[3] = 0.0f;
borderGradientPositions[2] = 1-0.03f;
borderGradientPositions[1] = 1-0.06f;
borderGradientPositions[0] = 1.0f;
```

3.2. Создайте массивы, в которых будут храниться позиции и цвета для радиального градиента. Они нужны для создания полупрозрачного «стеклянного купола», который покрывает Представление и создает эффект объема.

```
glassGradientColors = new int[5];
glassGradientPositions = new float[5];

int glassColor = 245;
glassGradientColors[4] = Color.argb(65, glassColor,
                                    glassColor, glassColor);
glassGradientColors[3] = Color.argb(100, glassColor,
                                    glassColor, glassColor);
glassGradientColors[2] = Color.argb(50, glassColor,
                                    glassColor, glassColor);
glassGradientColors[1] = Color.argb(0, glassColor,
                                    glassColor, glassColor);
glassGradientColors[0] = Color.argb(0, glassColor,
                                    glassColor, glassColor);
glassGradientPositions[4] = 1-0.0f;
glassGradientPositions[3] = 1-0.06f;
glassGradientPositions[2] = 1-0.10f;
```

```
glassGradientPositions[1] = 1-0.20f;
glassGradientPositions[0] = 1-1.0f;
```

3.3. В завершение получите цвета, которые понадобятся для создания линейных градиентов, чтобы отобразить небо и землю в автогоризонте.

```
  skyHorizonColorFrom = r.getColor(R.color.horizon_sky_from);
  skyHorizonColorTo = r.getColor(R.color.horizon_sky_to);

  groundHorizonColorFrom = r.getColor(R.color.horizon_ground_from);
  groundHorizonColorTo = r.getColor(R.color.horizon_ground_to);
}
```

4. Прежде чем приступать к рисованию циферблата, создайте новое перечисление, сохраняющее все значения для розы ветров.

```
private enum CompassDirection { N, NNE, NE, ENE,
                                E, ESE, SE, SSE,
                                S, SSW, SW, WSW,
                                W, WNW, NW, NNW }
```

Теперь необходимо полностью заменить метод onDraw. Начните с вычисления некоторых значений, зависящих от размера, включая центр Представления, радиус окружности компаса и прямоугольники, вмещающие в себя внешний (по контуру) и внутренний циферблаты.

```
@Override
protected void onDraw(Canvas canvas) {
```

5. Рассчитайте толщину внешнего кольца, исходя из размера шрифта, используемого для направляющих значений.

```
float ringWidth = textHeight + 4;
```

6. Вычислите высоту и ширину Представления, используя полученные значения для определения радиусов внутреннего и внешнего циферблатов и создания прямоугольных границ для каждого из них.

```
int height = getMeasuredHeight();
int width =getMeasuredWidth();

int px = width/2;
int py = height/2;
Point center = new Point(px, py);

int radius = Math.min(px, py)-2;

RectF boundingBox = new RectF(center.x - radius,
                              center.y - radius,
                              center.x + radius,
                              center.y + radius);

RectF innerBoundingBox = new RectF(center.x -   radius + ringWidth,
                                   center.y -   radius + ringWidth,
```

```
                                       center.x +    radius - ringWidth,
                                       center.y +    radius - ringWidth);

float innerRadius = innerBoundingBox.height()/2;
```

7. Поскольку вы уже получили размеры Представления, пора рисовать циферблаты.

   Начните с нижнего внешнего слоя (внешнего кольца), постепенно продвигаясь наверх и внутрь. Создайте новый шейдер RadialGradient, используя цвета и позиции, заданные в пункте 3.2. Передайте этот шейдер в новый объект Paint, прежде чем рисовать окружность.

```
RadialGradient borderGradient = new RadialGradient(px, py, radius,
  borderGradientColors, borderGradientPositions, TileMode.CLAMP);

Paint pgb = new Paint();
pgb.setShader(borderGradient);

Path outerRingPath = new Path();
outerRingPath.addOval(boundingBox, Direction.CW);

canvas.drawPath(outerRingPath, pgb);
```

8. Теперь нужно нарисовать автогоризонт. Разделите циферблат на две части, одна из которых будет означать небо, а другая — землю. Пропорции каждой из частей зависят от текущего параметра pitch.

   Начните с создания объектов Shader и Paint, они понадобятся для рисования неба и земли.

```
LinearGradient skyShader = new LinearGradient(center.x,
  innerBoundingBox.top, center.x, innerBoundingBox.bottom,
  skyHorizonColorFrom, skyHorizonColorTo, TileMode.CLAMP);

Paint skyPaint = new Paint();
skyPaint.setShader(skyShader);

LinearGradient groundShader = new LinearGradient(center.x,
  innerBoundingBox.top, center.x, innerBoundingBox.bottom,
  groundHorizonColorFrom, groundHorizonColorTo, TileMode.CLAMP);

Paint groundPaint = new Paint();
groundPaint.setShader(groundShader);
```

9. Нормализуйте значения pitch и roll, закрепив их в пределах ±90° и ±180° соответственно.

```
float tiltDegree = pitch;
while (tiltDegree > 90 || tiltDegree < -90)
{
  if (tiltDegree > 90) tiltDegree = -90 + (tiltDegree - 90);
    if (tiltDegree < -90) tiltDegree = 90 - (tiltDegree + 90);
}

float rollDegree = roll;
```

```
while (rollDegree > 180 || rollDegree < -180)
{
  if (rollDegree > 180) rollDegree = -180 + (rollDegree - 180);
    if (rollDegree < -180) rollDegree = 180 - (rollDegree + 180);
}
```

10. Создайте контуры, по которым станет закрашиваться каждый сегмент окружности (земля и небо). Пропорции сегментов должны соответствовать закрепленному параметру pitch.

```
Path skyPath = new Path();
skyPath.addArc(innerBoundingBox,
               -tiltDegree,
               (180 + (2 * tiltDegree)));
```

11. Поверните Холст вокруг центра в направлении, противоположном параметру roll, и закрасьте контуры для неба и земли, используя объекты, созданные в пункте 4.

```
canvas.rotate(-rollDegree, px, py);
canvas.drawOval(innerBoundingBox, groundPaint);
canvas.drawPath(skyPath, skyPaint);
canvas.drawPath(skyPath, markerPaint);
```

12. Приступайте к созданию циферблата. Начните с расчета начальной и конечной точек для горизонтальных меток автогоризонта.

```
int markWidth = radius / 3;
int startX = center.x - markWidth;
int endX = center.x + markWidth;
```

13. Чтобы горизонтальные значения были более простыми для восприятия, шкала параметра pitch должна всегда начинаться с текущего значения. Следующий код вычисляет положение границы между землей и небом на горизонтальной шкале.

```
double h = innerRadius*Math.cos(Math.toRadians(90-tiltDegree));
double justTiltY = center.y - h;
```

14. Вычислите количество пикселей, соответствующих каждому углу наклона.

```
float pxPerDegree = (innerBoundingBox.height()/2)/45f;
```

15. Переберите значения от 90° до –90°, исходя из текущего наклона, чтобы получить равномерную шкалу значений для параметра pitch.

```
for (int i = 90; i >= -90; i -= 10)
{
  double ypos = justTiltY + i*pxPerDegree;

  // Нарисуйте шкалу в рамках внутреннего циферблата.
  if ((ypos < (innerBoundingBox.top + textHeight)) ||
      (ypos > innerBoundingBox.bottom - textHeight))
```

```
    continue;

  // Нарисуйте линию и угол наклона для каждой метки шкалы.
  canvas.drawLine(startX, (float)ypos,
                  endX, (float)ypos,
                  markerPaint);
  int displayPos = (int)(tiltDegree - i);
  String displayString = String.valueOf(displayPos);
  float stringSizeWidth = textPaint.measureText(displayString);
  canvas.drawText(displayString,
                  (int)(center.x-stringSizeWidth/2),
                  (int)(ypos)+1,
                  textPaint);
}
```

16. Нарисуйте более толстую линию на границе земли и неба, изменив сперва толщину начертания объекта markerPaint (потом верните предыдущее значение).

```
markerPaint.setStrokeWidth(2);
canvas.drawLine(center.x - radius / 2,
                (float)justTiltY,
                center.x + radius / 2,
                (float)justTiltY,
                markerPaint);
markerPaint.setStrokeWidth(1);
```

17. Чтобы сделать точное значение параметра roll более простым для восприятия, нарисуйте стрелку и выведите текстовую строку, представляющую это значение.

    Создайте новый объект Path с использованием методов moveTo/lineTo, чтобы нарисовать стрелку, указывающую вверх. Нарисуйте контур и отобразите текстовую строку, которая показывает текущее значение параметра roll.

```
// Нарисуйте стрелку
Path rollArrow = new Path();
rollArrow.moveTo(center.x - 3, (int)innerBoundingBox.top + 14);
rollArrow.lineTo(center.x, (int)innerBoundingBox.top + 10);
rollArrow.moveTo(center.x + 3, innerBoundingBox.top + 14);
rollArrow.lineTo(center.x, innerBoundingBox.top + 10);
canvas.drawPath(rollArrow, markerPaint);
// Отобразите строку
String rollText = String.valueOf(rollDegree);
double rollTextWidth = textPaint.measureText(rollText);
canvas.drawText(rollText,
                (float)(center.x - rollTextWidth / 2),
                innerBoundingBox.top + textHeight + 2,
                textPaint);
```

18. Верните Холст обратно в вертикальное положение, чтобы нарисовать оставшиеся метки для циферблата.

```
canvas.restore();
```

19. Нарисуйте циферблат для обозначения углов отклонения, каждый раз поворачивая Холст на 10°, чтобы отобразить метку или значение. Закончив с циферблатом, верните Холст в изначальное вертикальное положение.

```
canvas.save();
canvas.rotate(180, center.x, center.y);
for (int i = -180; i < 180; i += 10)
{
  // Выводите цифровое значение каждые 30°
  if (i % 30 == 0) {
    String rollString = String.valueOf(i*-1);
    float rollStringWidth = textPaint.measureText(rollString);
    PointF rollStringCenter =
      new PointF(center.x-rollStringWidth/2,
                 innerBoundingBox.top+1+textHeight);
    canvas.drawText(rollString,
                    rollStringCenter.x, rollStringCenter.y,
                    textPaint);
  }
  // В ином случае, рисуйте отметку
  else {
    canvas.drawLine(center.x, (int)innerBoundingBox.top,
                    center.x, (int)innerBoundingBox.top + 5,
                    markerPaint);
  }

  canvas.rotate(10, center.x, center.y);
}
canvas.restore();
```

20. В завершение создайте циферблат, состоящий из направляющих меток вокруг внешнего кольца.

```
canvas.save();
canvas.rotate(-1*(bearing), px, py);

double increment = 22.5;

for (double i = 0; i < 360; i += increment) {
  CompassDirection cd = CompassDirection.values()
                        [(int)(i / 22.5)];
  String headString = cd.toString();

  float headStringWidth = textPaint.measureText(headString);
  PointF headStringCenter =
```

```
    new PointF(center.x - headStringWidth / 2,
               boundingBox.top + 1 + textHeight);

  if (i % increment == 0)
    canvas.drawText(headString,
                    headStringCenter.x, headStringCenter.y,
                    textPaint);
  else
    canvas.drawLine(center.x, (int)boundingBox.top,
                    center.x, (int)boundingBox.top + 3,
                    markerPaint);

  canvas.rotate((int)increment, center.x, center.y);
}
canvas.restore();
```

21. Закончив с циферблатом, добавьте завершающие штрихи. Начните с создания «стеклянного купола» поверх циферблата, сделав его похожим на механические часы. Применив массив со значениями для радиального градиента, определенный ранее, создайте новые объекты Shader и Paint. Используйте их, чтобы нарисовать окружность над внутренней поверхностью, как будто она покрыта стеклом.

```
RadialGradient glassShader =
  new RadialGradient(px, py, (int)innerRadius,
                     glassGradientColors,
                     glassGradientPositions,
                     TileMode.CLAMP);
Paint glassPaint = new Paint();
glassPaint.setShader(glassShader);

canvas.drawOval(innerBoundingBox, glassPaint);
```

22. Осталось нарисовать еще две окружности, чтобы очертить границы для внутренней и внешней поверхностей. Сделав это, восстановите Холст в вертикальном положении и закончите метод onDraw.

```
  // Нарисуйте внешнее кольцо
  canvas.drawOval(boundingBox, circlePaint);

  // Нарисуйте внутреннее кольцо
  circlePaint.setStrokeWidth(2);
  canvas.drawOval(innerBoundingBox, circlePaint);

  canvas.restore();
}
```

«Оживляем» Наложения, размещенные поверх MapView

В главе 8 вы научились с помощью Наложений добавлять слои с аннотациями поверх элементов MapView. При этом использовался тот же класс Canvas, что и при создании нестандартных Представлений. Поэтому применив все продвинутые возможности, описанные ранее в этом разделе, можно улучшить и карточные Наложения. Это значит, что вы можете задействовать

любые методы для рисования, прозрачность, шейдеры, цветовые маски и эффекты, предоставляемые графическим фреймворком Android, чтобы создавать визуально богатые и насыщенные Наложения.

За взаимодействие с сенсорным экраном в MapView отвечает каждое отдельное его Наложение. Чтобы реагировать на нажатия внутри Наложения, переопределите событие onTap.

В листинге 15.22 показана реализация метода onTap, который в качестве параметров принимает координаты нажатия на карте, а также соответствующий объект MapView.

Листинг 15.22. Обработка событий, связанных с нажатиями внутри Наложений

```
@Override
public boolean onTap(GeoPoint point, MapView map) {
  // Получите проекцию для преобразования экранных координат (в обе
  // стороны)
  Projection projection = map.getProjection();

  // Верните true, если мы обработали это событие
  return [ ... проверка на нажатие пройдена ... ];
}
```

Объект MapView может быть использован при нажатии на экране для получения проекции карты. Применяя эту проекцию в сочетании с параметром GeoPoint, можно определить позицию на экране, соответствующую географическим координатам.

Метод onTap, принадлежащий производным от Overlay классам, должен возвращать значение true, если он действительно обрабатывает нажатие (и false — если нет). Когда ни одно из Наложений, принадлежащих элементу MapView, не вернуло true, событие нажатия будет обработано самим MapView или Активностью.

Знакомство с SurfaceView

В обычных условиях Представления из вашего приложения рисуются в едином графическом потоке. Этот главный программный поток также отвечает за все взаимодействия с пользователем (например, нажатие кнопок или ввод текста).

В главе 9 вы научились перемещать блокирующие операции в фоновые потоки. К сожалению, вы не можете сделать то же самое с методом onDraw из своего Представления, так как изменение элементов графического интерфейса из фонового потока явно запрещено.

Если вам необходимо быстро обновить пользовательский интерфейс Представления или если код для отрисовки слишком долго блокирует графический поток, применение класса SurfaceView может стать выходом из положения. В основе его объект Surface, а не Canvas. Это важно, потому

как Surface поддерживает рисование из фоновых потоков. Данное отличие особенно полезно для ресурсоемких операций или быстрых обновлений, а также когда необходимо обеспечить высокую частоту изменения кадров (использование трехмерной графики, создание игр или предпросмотр видеопотока с камеры в режиме реального времени, как продемонстрировано в главе 11).

Возможность рисовать вне зависимости от графического потока ведет к повышенному потреблению памяти. Таким образом, хоть это и полезный (а иногда просто необходимый) способ создания нестандартных Представлений, будьте осторожны, используя его.

В каких случаях нужно использовать SurfaceView

SurfaceView используется точно таким же образом, как любые производные от View классы. Вы можете применять анимацию и размещать их внутри разметки так же, как и другие Представления.

Объект Surface, который лежит в основе SurfaceView, поддерживает рисование с помощью большинства стандартных методов Canvas, описанных ранее, а также может использовать все возможности библиотеки OpenGL ES.

Применяя OpenGL, вы можете рисовать на Surface любые поддерживаемые двумерные или трехмерные объекты, получая при этом все выгоды от аппаратного ускорения (если таковое имеется). Таким образом, вы значительно повышаете производительность, если сравнивать с теми же операциями, выполненными на двумерном Canvas.

Объекты SurfaceView особенно пригодятся для отображения динамических трехмерных изображений, к примеру, в интерактивных играх, их можно назвать лучшим выбором для отображения предварительного просмотра видеопотоков с камеры в режиме реального времени.

Создание нового объекта SurfaceView

Чтобы создать данный тип Представления, наследуйте класс SurfaceView и реализуйте интерфейс SurfaceHolder.Callback, описывающий функцию обратного вызова. Он уведомляет Представление о том, что исходный объект Surface был создан/уничтожен/модифицирован и передает в объект SurfaceHolder ссылку, содержащую допустимый экземпляр Surface.

Типичный шаблон проектирования SurfaceView предусматривает классы, производные от Thread, которые принимают ссылку на текущий объект SurfaceHolder и немедленно его обновляют.

В листинге 15.23 показана реализация SurfaceView, рисование с помощью объекта Canvas. Новые производные от Thread классы создаются внутри элемента SurfaceView, и все обновления пользовательского интерфейса происходят в этом новом классе.

Листинг 15.23. Каркас для реализации класса SurfaceView

```
import android.content.Context;
import android.graphics.Canvas;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MySurfaceView extends SurfaceView implements
  SurfaceHolder.Callback {

  private SurfaceHolder holder;
  private MySurfaceViewThread mySurfaceViewThread;
  private boolean hasSurface;

  MySurfaceView(Context context) {
    super(context);
    init();
  }

  private void init() {
    // Создайте новый объект SurfaceHolder  и определите данный
    // класс в качестве его функции обратного вызова.
    holder = getHolder();
    holder.addCallback(this);
    hasSurface = false;
  }

  public void resume() {
    // Создайте и запустите поток для обновления UI.
    if (mySurfaceViewThread == null) {
      mySurfaceViewThread = new MySurfaceViewThread();

      if (hasSurface == true)
        mySurfaceViewThread.start();
    }
  }

  public void pause() {
    // Уничтожьте поток для обновления UI.
    if (mySurfaceViewThread != null) {
      mySurfaceViewThread.requestExitAndWait();
      mySurfaceViewThread = null;
    }
  }

  public void surfaceCreated(SurfaceHolder holder) {
    hasSurface = true;
    if (mySurfaceViewThread != null)
      mySurfaceViewThread.start();
  }

  public void surfaceDestroyed(SurfaceHolder holder) {
    hasSurface = false;
    pause();
  }

  public void surfaceChanged(SurfaceHolder holder, int format,
```

Продолжение ⇗

Листинг 15.23 (продолжение)

```
                              int w, int h) {
      if (mySurfaceViewThread != null)
        mySurfaceViewThread.onWindowResize(w, h);
    }

    class MySurfaceViewThread extends Thread {
      private boolean done;

      MySurfaceViewThread() {
        super();
        done = false;
      }

      @Override
      public void run() {
        SurfaceHolder surfaceHolder = holder;

        // Повторяйте цикл перерисовки, пока поток не будет остановлен.
        while (!done) {
          // Заблокируйте Surface и верните Canvas, на котором можно
          // рисовать.
          Canvas canvas = surfaceHolder.lockCanvas();
          // TODO: Процесс рисования на объекте Canvas!
          // Разблокируйте Canvas и отобразите текущее изображение.
          surfaceHolder.unlockCanvasAndPost(canvas);
        }
      }

      public void requestExitAndWait() {
        // Пометьте этот поток как завершенный и соедините
        // его с главным потоком приложения.
        done = true;
        try {
          join();
        } catch (InterruptedException ex) { }
      }

      public void onWindowResize(int w, int h) {
        // Обработайте изменение доступных размеров поверхности.
      }
    }
  }
```

Создание трехмерных элементов управления с помощью SurfaceView

Android содержит фреймворк с полной поддержкой отображения трехмерных сцен с помощью OpenGL ES, включая аппаратное ускорение (на устройствах, где это доступно). Объект SurfaceView предоставляет поверхность, на которой вы можете отображать сцены, используя функции OpenGL.

OpenGL, как правило, используется в настольных приложениях для создания динамических трехмерных интерфейсов и анимации. Устройства

с ограниченными ресурсами не могут обрабатывать такой объем полигонов, как настольные ПК и игровые консоли с дискретными графическими процессорами. При разработке своих приложений помните, что вся нагрузка, создаваемая трехмерными сценами на вашем SurfaceView, может «лечь на плечи» центрального процессора, поэтому пытайтесь максимально снизить количество полигонов, выводимых на экран, и частоту, с которой они обновляются.

Создание клона игры Doom для Android явно выходит за рамки этой книги, поэтому, если хотите, можете сами проверить, на что способен мобильный трехмерный пользовательский интерфейс. Ознакомьтесь с демонстрационным проектом GLSurfaceView, входящим в состав SDK, чтобы оценить в действии возможности фреймворка OpenGL ES.

Создание интерактивных элементов управления

Многие пользователи смартфонов на личном опыте сталкивались с проблемами, связанными с проектированием интуитивных пользовательских интерфейсов для мобильных устройств. Сенсорные экраны не новинка в мире мобильных технологий, но лишь совсем недавно интерфейсы начали оптимизировать для управления пальцами, а не стилусами.

Полноценные физические клавиатуры компактных размеров (выдвижные или откидные) также стали обычным явлением.

Как открытая платформа Android обязан быть доступным для широкого спектра устройств, поддерживающих различные технологии ввода данных, включая сенсорные экраны, манипуляторы типа D-pad, трекболы и клавиатуры.

Ваша задача как разработчика заключается в создании интуитивных пользовательских интерфейсов, которые умеют работать с большинством устройств ввода, порождая как можно меньше аппаратных зависимостей.

Подходы, описанные в этом разделе, демонстрируют, как с помощью нижеприведенных обработчиков внутри Представлений и Активностей отслеживать пользовательский ввод (и реагировать на него), связанный с нажатиями клавиш, событиями трекбола и сенсорного экрана:

- onKeyDown — вызывается при нажатии любой аппаратной клавиши;
- onKeyUp — вызывается при отпускании любой аппаратной клавиши;
- onTrackballEvent — срабатывает при движениях трекбола;
- onTouchEvent — обработчик событий сенсорного экрана, срабатывает при касании, убирании пальца и при перетаскивании.

Использование сенсорного экрана

Мобильные сенсорные экраны существуют со времен Apple Newton и Palm Pilot, хотя удобство их использования вызывало смешанные чувства. Недавно эта технология обрела вторую жизнь вместе с появлением таких устройств, как Nintendo DS и Apple iPhone, принесших множество инноваций.

Современные мобильные устройства нельзя представить без пальцевого ввода — они разрабатываются с расчетом, что пользователи будут прикасаться к сенсорныму экрану пальцем, а не стилусом.

Пальцевый ввод менее точный и часто основывается на движении, а не на простом прикосновении. Стандартные приложения в Android широко используют сенсорные интерфейсы, рассчитанные на пальцы, помимо всего прочего, они предусматривают использование жестов «перетаскивания» для прокрутки списков или выполнения других действий.

Чтобы создать Представление или Активность, которые могут взаимодействовать с сенсорным экраном, переопределите обработчик onTouchEvent:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  return super.onTouchEvent(event);
}
```

Верните true, если вы обработали нажатие на экран. В ином случае верните false, чтобы передавать события по цепочке от Представления до Активности, пока оно не будет успешно обработано.

Обработка одиночных и множественных касаний. onTouchEvent срабатывает в трех случаях: когда пользователь прикасается к экрану, изменяет позицию касания и отпускает палец. В Android 2.0 (API level 5) появилась поддержка платформы для обработки произвольного количества одновременных событий, связанных с сенсорным экраном. Для каждого касания выделяется отдельный точечный идентификатор, ссылающийся на параметр MotionEvent.

ПРИМЕЧАНИЕ

Не все сенсорные экраны уведомляют о нескольких одновременных нажатиях. Если аппаратное обеспечение не поддерживает множественные касания, Android возвращает одиночное событие.

Вызовите метод getAction из параметра MotionEvent, чтобы определить тип события, которое запустило обработчик. Независимо от того, поддерживает устройство множественные касания или нет, вы все равно можете использовать константы ACTION_UP/DOWN/MOVE/CANCEL/OUTSIDE для получения типа события, как показано в листинге 15.24.

Листинг 15.24. Обработка одиночных (или первых) касаний

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  int action = event.getAction();
  switch (action) {
    case (MotionEvent.ACTION_DOWN)   : // Нажатие на сенсорном экране
                                       break;
    case (MotionEvent.ACTION_UP)     : // Завершение нажатия
                                       break;
    case (MotionEvent.ACTION_MOVE)   : // Контакт был перемещен по экрану
                                       break;
    case (MotionEvent.ACTION_CANCEL) : // Касание было отменено
                                       break;
    case (MotionEvent.ACTION_OUTSIDE): // Произошло движение за пределами
                                       // границ экранного элемента,
                                       // который отслеживается
                                       break;
  }
  return super.onTouchEvent(event);
}
```

Чтобы следить за касаниями в разных точках, необходимо использовать константы MotionEvent.ACTION_MASK и MotionEvent.ACTION_POINTER_ID_MASK для определения типа события (будь то ACTION_POINTER_DOWN или ACTION_POINTER_UP) и идентификатора контакта, который его вызвал, соответственно. На примере листинга 15.25 вызовите метод getPointerCount, чтобы определить наличие множественных касаний.

Листинг 15.25. Обработка множественных касаний

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  int action = event.getAction();

  if (event.getPointerCount() > 1) {
    int actionPointerId = action & MotionEvent.ACTION_POINTER_ID_MASK;
    int actionEvent = action & MotionEvent.ACTION_MASK;
    // Какие-либо действия с идентификатором контакта и событием.
  }
  return super.onTouchEvent(event);
}
```

Объект MotionEvent содержит координаты текущего контакта с экраном. Вы можете получить их с помощью методов getX и getY. Возвращенные значения — относительные для соответствующего Представления или Активности.

В случае с множественными касаниями объект MotionEvent включает текущую позицию каждого контакта. Чтобы найти координаты заданного контакта, передавайте его индекс в методы getX и getY. Обратите внимание, что этот индекс не имеет никакого отношения к идентификатору

контакта. Чтобы получить индекс заданного контакта, используйте метод findPointerIndex, передавая ему ID соответствующего касания, как показано в листинге 15.26.

Листинг 15.26. Получение координат соприкосновения с экраном

```
int xPos = -1; int yPos = -1;

if (event.getPointerCount() > 1) {
  int actionPointerId = action & MotionEvent.ACTION_POINTER_ID_MASK;
  int actionEvent = action & MotionEvent.ACTION_MASK;

  int pointerIndex = findPointerIndex(actionPointerId);
  xPos = (int)event.getX(pointerIndex);
  yPos = (int)event.getY(pointerIndex);
}
else {
  // Одиночное касание.
  xPos = (int)event.getX();
  yPos = (int)event.getY();
}
```

Параметр MotionEvent содержит степень давления, которое было применено к экрану. Получить его можно с помощью метода getPressure, который возвращает значение в пределах между 0 (без давления) и 1 (нормальное давление).

ПРИМЕЧАНИЕ

В зависимости от калибровки аппаратного обеспечения это значение может быть больше 1.

Наконец, вы также можете определить нормализованный размер текущей области соприкосновения, используя метод getSize. Он возвращает значения в диапазоне между 0 и 1, где 0 обозначает очень точное измерение, а 1 указывает на возможное «толстое» прикосновение, при котором пользователь, вероятно, вовсе не пытался ничего нажать.

Отслеживание движений. Каждый раз при изменении позиции текущего прикосновения, его давления или размера срабатывает новый обработчик onTouchEvent с действием ACTION_MOVE.

Как и ранее упоминавшиеся поля, параметр MotionEvent может содержать значения, описывающие все движения, обнаруженные между текущим и предыдущим срабатыванием обработчика onTouchEvent. Это позволяет операционной системе добавлять в буфер быстрые изменения движения, чтобы иметь возможность отслеживать эти данные в мельчайших подробностях.

Вы можете найти размер истории изменений, вызвав метод getHistorySize, который вернет количество позиций, доступных для текущего события. Затем можно узнать продолжительность, давление, размер и позицию каждого

события из буфера, используя набор методов getHistorical* и передавая индексы позиций, как показано в листинге 15.27. Обратите внимание, что, как и в случае с методами getX и getY, описанными ранее, вы можете передать индекс контакта для получения данных о касаниях из нескольких точек.

Листинг 15.27. Получение истории изменения значений для событий касания

```
int historySize = event.getHistorySize();
long time = event.getHistoricalEventTime(i);

if (event.getPointerCount() > 1) {
  int actionPointerId = action & MotionEvent.ACTION_POINTER_ID_MASK;
  int pointerIndex = findPointerIndex(actionPointerId);
  for (int i = 0; i < historySize; i++) {
    float pressure = event.getHistoricalPressure(pointerIndex, i);
    float x = event.getHistoricalX(pointerIndex, i);
    float y = event.getHistoricalY(pointerIndex, i);
    float size = event.getHistoricalSize(pointerIndex, i);
    // TODO: Какие-либо действия с каждым контактом
  }
}
else {
  for (int i = 0; i < historySize; i++) {
    float pressure = event.getHistoricalPressure(i);
    float x = event.getHistoricalX(i);
    float y = event.getHistoricalY(i);
    float size = event.getHistoricalSize(i);
    // TODO: Какие-либо действия с каждым контактом
  }
}
```

На деле для реагирования на движения необходимо обрабатывать каждое событие из истории, следующее за текущими значениями из объекта MotionEvent, как продемонстрировано в листинге 15.28.

Листинг 15.28. Обработка касательных движений по экрану

```
@Override
public boolean onTouchEvent(MotionEvent event) {

  int action = event.getAction();

  switch (action) {
    case (MotionEvent.ACTION_MOVE)
    {
      int historySize = event.getHistorySize();
      for (int i = 0; i < historySize; i++) {
        float x = event.getHistoricalX(i);
        float y = event.getHistoricalY(i);
        processMovement(x, y);
      }

      float x = event.getX();
      float y = event.getY();
```

Продолжение ⇨

Листинг 15.28 (продолжение)

```
        processMovement(x, y);

        return true;
      }
    }

    return super.onTouchEvent(event);
    }

    private void processMovement(float _x, float _y) {
    // Todo: Обработка движения.
  }
```

Использование интерфейса OnTouchListener. Вы можете отслеживать касания без наследования ранее созданных Представлений, просто добавив OnTouchListener к любому объекту View, используя метод setOnTouchListener. В листинге 15.29 показано, как назначить новую реализацию интерфейса OnTouchListener для существующего Представления внутри Активности.

Листинг 15.29. Назначение OnTouchListener для существующего Представления

```
myView.setOnTouchListener(new OnTouchListener() {
  public boolean onTouch(View _view, MotionEvent _event) {
    // TODO Реакция на движение
  return false;
  }
});
```

Использование клавиатуры, аппаратных клавиш и манипулятора D-pad

Событие, сигнализирующее о нажатии любой аппаратной кнопки обрабатывается методами onKeyDown и onKeyUp, принадлежащими Активности или выделенному Представлению. Это касается клавиатуры, манипулятора D-pad, клавиш для изменения звука, навигации, вызова и отбоя. Единственное исключение — кнопка home, взаимодействие с которой ограничено, чтобы пользователь всегда мог выйти из приложения.

Чтобы ваши Представления и Активности реагировали на нажатия клавиш, переопределите обработчики событий onKeyUp и onKeyDown, как показано в листинге 15.30.

Листинг 15.30. Обработка нажатий клавиш

```
@Override
public boolean onKeyDown(int _keyCode, KeyEvent _event) {
  // Обработайте нажатие, верните true, если обработка выполнена
  return false;
}

@Override
```

```
public boolean onKeyUp(int _keyCode, KeyEvent _event) {
  // Обработайте отпускание клавиши, верните true, если обработка
выполнена
  return false;
}
```

Параметр _keyCode содержит код клавиши, которая была нажата; сравнивайте его со статическими кодами клавиш, хранящимися в классе KeyEvent, чтобы выполнять соответств ующую обработку.

Параметр KeyEvent также включает в себя несколько методов: isAltPressed, isShiftPressed и isSymPressed, определяющих, были ли нажаты функциональные клавиши, такие как **Alt**, **Shift** или **Sym**. Статический метод isModifierKey принимает _keyCode и определяет, является ли нажатая клавиша модификатором.

Использование интерфейса OnKeyListener

Чтобы среагировать на нажатие клавиши внутри существующего Представления из Активности, реализуйте интерфейс OnKeyListener и назначьте его для объекта View, используя метод setOnKeyListener. Вместо того, чтобы реализовывать отдельные методы для событий нажатия и отпускания клавиш, OnKeyListener использует единое событие onKey. Данный механизм показан в листинге 15.31.

Листинг 15.31. Реализация OnKeyListener внутри Активности

```
myView.setOnKeyListener(new OnKeyListener() {
  public boolean onKey(View v, int keyCode, KeyEvent event)
  {
    // TODO Обработайте нажатие клавиши, верните true, если
    // обработка выполнена
    return false;
  }
});
```

Используйте параметр keyCode для получения клавиши, которая была нажата. Параметр KeyEvent нужен для распознавания типа события (нажатие представлено константой ACTION_DOWN, а отпускание — ACTION_UP).

Использование трекбола

Многие мобильные устройства предоставляют трекбол в качестве удобной альтернативы (или дополнения) сенсорному экрану и манипулятору D-pad. События, посылаемые трекболом, обрабатываются путем переопределения метода onTrackballEvent внутри ваших Представлений или Активностей.

Как и в случае с касаниями, данные о движении трекбола содержатся в параметре MotionEvent. При этом MotionEvent включает относительное

движение трекбола с момента поступления последнего события; значения нормализуются таким образом, что 1 означает движение, эквивалентное нажатию на манипулятор D-pad.

Изменения по вертикали могут быть получены с помощью метода getY, а горизонтальная прокрутка доступна через метод getX, как показано в листинге 15.32.

Листинг 15.32. Отслеживание событий, посылаемых трекболом

```
@Override
public boolean onTrackballEvent(MotionEvent _event) {
  float vertical = _event.getY();
  float horizontal = _event.getX();
  // TODO Обработка движений трекбола.
  return false;
}
```

Резюме

Эта завершающая глава охватывает некоторые из наиболее сложных возможностей платформы Android. Вы узнали больше о механизмах безопасности в Android, в частности, рассмотрели систему полномочий, которая использовалась для предоставления доступа к Источникам данных, Сервисам, Активностям, Приемникам широковещательных намерений и к самим Намерениям.

Вы изучили возможности межпроцессного взаимодействия, используя AIDL (Android Interface Definition Language) для создания насыщенных интерфейсов между компонентами приложения.

Большинство разделов посвящены классу Canvas, в частности, рассмотрены более сложные возможности, предоставляемые библиотеками для рисования двумерной графики. Эта часть главы познакомила с процессом рисования доступных графических примитивов, а также продемонстрировала возможности класса Paint.

Вы научились использовать полупрозрачность, создавать градиентные шейдеры, рассмотрели фильтры для масок, цветовые фильтры, конкурные эффекты. Узнали, как применить аппаратное ускорение для двумерных Представлений, опробовали некоторые общепринятые подходы к рисованию с помощью Canvas. Познакомились графическим элементом классом SurfaceView, с помощью которого можно отображать графику на экране прямо из фонового потока. Далее последовало введение в трехмерную графику с использованием фреймворка OpenGL ES, а также демонстрация применения SurfaceView для предварительного просмотра живых видеопотоков с камеры.

В завершение вы узнали, как взаимодействовать с пользователем посредством своих Активностей и Представлений, отслеживая и интерпретируя события, поступающие из сенсорного экрана, трекбола и аппаратных клавиш.

Вы также изучили:

- как использовать объекты WakeLock, чтобы не дать устройству уйти в режим ожидания;
- добавлять голосовой ввод в свои приложения, используя библиотеки для преобразования текста в речь;
- использовать Глобальную сеть в качестве источника данных или промежуточного звена для их обработки, чтобы сделать свои приложения информационно насыщенными, при этом сохраняя их легковесность;
- анимировать отдельные Представления и их группы с помощью анимации, создаваемой путем расчета промежуточных кадров;
- создавать пошаговую анимацию;
- применять графические примитивы, доступные для рисования с помощью Canvas;
- выжать максимум из объекта Paint, используя прозрачность, шейдеры, фильтры масок, цветовые фильтры и контурные эффекты;
- рисовать на Холсте;
- применять аппаратное ускорение при рисовании двумерных сцен.

Алфавитный указатель

#

.aidl 600
.apk 48 55 78
.class 47, 79
.dex 41, 47, 55
.xml 106
/data/data/<имя_пакета>/databases 279
?android: 110
@ 109, 110
@echo 506
[get/set]ColorEffect 476
[get/set]FlashMode 476
[get/set]FocusMode 476
[get/set]SceneMode 475
[get/set]WhiteBalance 476
<activity> 92
<animation-list> 616
<application> 92, 117, 344
<color> 101, 161
<corners> 161
<dimen> 102
<gradient> 163
<intent-filter> 92, 413, 491
<item> 167, 186
<layer-list> 167
<manifest> 88
<menu> 186
<meta-data> 448
<padding> 162
<permission> 591
<rotate> 114
<scale> 165
<service> 456
<shape> 161
<solid> 162
<string> 101
<stroke> 162
<style> 102
<supports-screens> 171
<uses-features> 94
<uses-permission> 326, 340, 506, 590
<uses-sdk> 94
<wallpaper> 455
«Здравствуй, мир» *см.* Hello World
3G 27, 63, 234, 555

A

AAPT *см.* Инструмент для создания пакетов
accept 538
acceptMatch 218
ACCESS_COARSE_LOCATION 324, 497
accuracy 568
action 203
ACTION_ANSWER 201
ACTION_BOOT_COMPLETED 224
ACTION_CALL 201
ACTION_CAMERA_BUTTON 224
ACTION_DATE_CHANGED 224
ACTION_DELETE 202
ACTION_DIAL 202
ACTION_DISCOVERY_FINISHED 536
ACTION_DISCOVERY_STARTED 536
ACTION_DOWN 645
ACTION_EDIT 202
ACTION_FOUND 536
ACTION_IMAGE_CAPTURE 473

ACTION_INSERT 202
ACTION_MEDIA_BUTTON 224
ACTION_MEDIA_EJECT 224
ACTION_MEDIA_MOUNTED 224
ACTION_MEDIA_UNMOUNTED 224
ACTION_MOVE 642
ACTION_NEW_OUTGOING_CALL 224
ACTION_PICK 202, 207, 210, 211
ACTION_REQUEST_DISCOVERABLE 534, 539
ACTION_SCAN_MODE_CHANGED 535
ACTION_SCREEN_OFF 224
ACTION_SCREEN_ON 224
ACTION_SEARCH 202
ACTION_SEND 202, 501
ACTION_SENDTO 202, 501
ACTION_STATE_CHANGED 532
ACTION_TIME_CHANGED 224
ACTION_TIMEZONE_CHANGED 224
ACTION_UP 640, 645
ACTION_VIDEO_CAPTURE 468
ACTION_VIEW 203
ACTION_WEB_SEARCH 203
Activity *см.* Активности
Activity.getPreferences() 271
Activity.RESULT_CANCELED 199
Activity.RESULT_OK 199, 503
activityCreator 79
Adapter *см.* Адаптер
AdapterView 187, 226
ADB *см.* Android Debug Bridge
add_new 111
addIntentOptions 214
addNetwork 563
address 501
addSubMenu 183
addView 137
ADT *см.* Android Developer Tool
AIDL *см.* Язык описания интерфейсов в Android
Alarm *см.* Сигнализация
AlarmManager 409, 431
AlertDialog 237
ALL 217
alpha 104, 105
ALTERNATIVE 204, 214
Amazon Web Services 603
AnalogClock 421
Android Debug Bridge 78, 83
    передача SMS 502
Android Developer Tool 17, 47
    обновление 50
    установка 48
Android Interface Definition Language *см.* Язык описания интерфейсов в Android
Android Open Source Project 26
android.app.searchable 447
android.net.* 235
android.provider.Settings 267, 311
android.provider.Telephony.SMS_RECEIVED 505, 507
android.settings.* 267
android.telephony.gsm 502
android:apikey 345, 360
android:autoLink 217
android:configChanges 114
android:defaultValue 265
android:host 204
android:ID 326
android:key 265
android:label 215
android:mimetype 205
android:name 92 203
android:path 205
android:permission 93
android:port 205
android:scheme 205
android:summary 265
android:title 265
AnimationDrawable 616
AnimationListener 607

AnimationResources 109
anyDensity 91
AOSP *см.* Android Open Source Project
App Engine 603
Apple iPhone 23, 571, 640
application 95
Application 92, 116
  наследование 116
  переопределение событий 117
APPWIDGET_CONFIGURE 432
appwidget-provider 422
AppWidgetProvider 423
AppWidgets *см.* Виджеты
ArrayAdapter 71, 76, 144, 189, 226, 227, 228, 229, 232, 243, 246, 251, 258, 293, 295, 511, 549
ArrayList 76, 229, 251, 597
AsyncTask 367, 384
  запуск асинхронной задачи 387
  создание асинхронной задачи 385
AT_MOST 151
AudioRecord 460, 482
  запись звука 483
AudioTrack 460, 482
  воспроизведение звука 484
AutoFocusCallback 477
autolink 250
AutoResponder 520
Available Packages 46
AVD *см.* Виртуальные устройства в Android
AvoidXfermode 624

B

bindService 382
Bitmap 617
BitmapShader 620
BLUETOOTH 529
Bluetooth 16, 528
  библиотеки 29, 529
  выбор 540
  доступ 529
  обнаружение 533
  передача данных 545
  поиск 535
  связь 539
  серверный сокет 538, 542
  сканирование 533
  управление 530
BluetoothAdapter 529
BluetoothDevice 529, 536, 540
BluetoothDevice.EXTRA_NAME 536
BluetoothServerSocket 529, 538
BluetoothSocket 529, 533, 538
BluetoothSocketListener 552
Bluetooth-адаптер 528
  getBondedDevices 541
BlurMaskFilter 622
BROADCAST_STICKY 223
BroadcastReceiver 87, 220, 332, 523
BROWSABLE 204
Browser 311
BSSID 560
bulkInsert 300
Bundle 126, 253, 272
Button 134, 421

C

C 13
C++ 13
calculateHeight 151
calculateOrientation 585
calculateWidth 151
CALL_PHONE 492
CallLog 311
CAMERA 475
Camera *см.* Камера
Camera.Parameters 475
cancel 111, 401
cancelAdd 192
cancelDiscovery 536
Canvas *см.* Холст

category 204
CATEGORY_ALTERNATIVE 215
CATEGORY_SELECTED_ALTERNATIVE 215
CDMA 492
CellLocation 498
centerX 163
centerY 153
CharacterPickerDialog 239
CharSequence 597
CheckBox 134, 180, 519
CheckBoxPreference 266
Chronometer 421
circlePaint 158
CLAMP 621
ClearableEditText 146
color 162
ColorDrawable 160
ColorFilter 505–506
ColorMatrixColorFilter 622
commit 254
Commmon Language Runtime 85
CommonDataKinds 315
CompassView, создание 154
ComposePathEffect 623
ComposeShader 620
Configuration 115
configureBluetooth 547
connect 481, 542
ConnectivityManager 555
ConnectivityManager 555
ContactPicker 208
Contacts 311
ContactsContract 311, 313, 317
ContactsContract.Contacts.CONTENT_FILTER_URI 315
ContactsContract.Data 315
ContactsContract.PhoneLookup.CONTENT_FILTER_URI 316
ContactsContract.StatusUpdates 317
Content Providers *см.* Источники данных
ContentResolver 296, 299, 314, 482
    вставка 300
    обновление 301
    удаление 301
content://contacts/people 202, 208
CONTENT_URI 295, 299
contentIntent 397
ContentPickerTester 210
ContentProvider 61, 87, 276, 295, 303
ContentSlider 608
ContentValues 280, 288, 482
    put 288
contentView 397, 400
Context *см.* Контекст
Context.MODE_APPEND 275
Context.MODE_WORLD_READABLE 276
Context.Mode_WORLD_WRITEABLE 276
ContextMenu 184
ContextMenuInfo 191
CornerPathEffect 623
create, Медиапроигрыватель 462
CREATE_LIVE_FOLDER 443
createItem 356
createRfcommSocketToServiceRecord 529, 542
Criteria 323, 328
Cursor *см.* Курсор
CursorFactory 284

D

Dalvik 25, 27, 38, 40
    Служба для отслеживания процесса отладки в Dalvik 44, 48, 78, 82
DashPathEffect 623
data 204
Data 313
DatePickerDialog 239
DDMS *см.* Dalvik, Служба для отслеживания процесса отладки в Dalvik
debuggable 92
DEFAULT 204, 447

defaults 404
delete 287, 297, 300, 303
deleteFile 276
deliveryIntent 503
Dialog 236, 239
direction 499
DiscretePathEffect 623
divideMessage 504
doInBackground 386
dp 102
draw 348
drawable 56
Drawable 98, 103, 104, 106, 114, 160, 173, 181, 428, 615, 625
drawArc 617
drawARGB/drawRGB/drawColor 617
drawBitmap 617
drawBitmapMesh 617
drawCircle 158, 617
drawLine(s) 618
drawOval 618
drawPaint 618
drawPath 618
drawPicture 618
drawPosText 618
drawRect 618
drawRoundRect 618
drawText 618
drawTextOnPath 618
drawVertices 618
Droid 170
Duration 105
Dx 79

E

Eclipse 38, 44, 47
 работа с плагином 47
 разработка в среде 47
EDGE 27, 63
EditText 75, 134, 145, 421, 518
EditTextPreference 266
EFFECT_* 476
ELAPSED_REALTIME 410, 431
ELAPSED_REALTIME_WAKEUP 410
EMAIL_ADDRESSES 217
EmbossMaskFilter 622
Emergency Responder, приложение:
 EmergencyResponder 508, 511, 513, 514, 516, 518, 525, 526
 автоматизация 518
 создание 508
 endColor 163
 enum 612
EXACTLY 151
Exchangeable Image File Format *см.* Камера, метаданные EXIF
execSQl 285
execute 385, 387
EXIF *см.* Камера, метаданные EXIF
ExifInterface 480
ExpandableListActivity 128
EXTRA_APPWIDGET_ID 433
EXTRA_BCC 202
EXTRA_BSSID 560
EXTRA_CC 202
EXTRA_DISCOVERABLE_DURATION 534
EXTRA_EMAIL 202
EXTRA_EXTRA_INFO 559
EXTRA_IS_FAILOVER 559
EXTRA_LANGUAGE 487
EXTRA_MAXRESULTS 487
EXTRA_NETWORK_INFO 559
EXTRA_NO_CONNECTIVITY 559
EXTRA_OTHER_NETWORK_INFO 559
EXTRA_OUTPUT 468
EXTRA_PHONE_NUMBER 224
EXTRA_PREVIOUS_STATE 532
EXTRA_PROMPT 487
EXTRA_REASON 559
EXTRA_RESULTS 487

EXTRA_STATE 532
EXTRA_STREAM 202
EXTRA_SUBJECT 202
EXTRA_TEXT 202
EXTRA_VIDEO_QUALITY 468

F

file:// 462
fileList 208
FILL 618
fill_parent 137, 172
fillAfter 105
fillBefore 105
findItem 180, 182
findViewById 57, 75, 133
FingerPaint 622
finish 197, 199
flag 372
FLAG_INSISTENT 407
FLAG_ONGOING_EVENT 407
FLAG_SHOW_LIGHTS 407
FLASH_MODE_* 476
FOCUS_MODE_* 476
fontScale 115
FrameLayout 135, 421
fromDegrees 105, 165
FULL_WAKE_LOCK 593

G

GADGET 204
Gallery 136
gData 603
Geocoder 320, 333
GeoPoint 343, 359, 635
get<read/writ>ableDatabase 285
get<тип> 207, 254, 258, 287
getAction 207, 540
getActiveNetworkInfo 557
getAttribute 480
getBestProvider 323
getBondedDevices 541
getColumnIndexOrThrow 281
getColumnName 281
getColumnNames 281
getConfiguredNetworks 562
getContentResolver 299
getCount 281
getCurrentPosition 466
getData 207
getDataActivity 493
getDataState 493
getDefault 487
getDefaultSensor 567
getDrawable 109
getFrame 467
getFromLocation 334
getFromLocationName 335
getHistorical* 643
getHistorySize 642
getInputStream 544
getIntent 207
getItem 227
getLastKnownLocation 322, 326
updateWithNewLocation 326
getMode 151
getNetworkInfo 557
getNetworkPreference 558
getOperator* 498
getOutputStream 544
getParameters 475
getPhoneType 494
getPointerCount 641
getPosition 281
getProjection 348
getProvider 322
getProviders 323
getReadableDatabase 284
getResources 108
getRoaming 498
getScanMode 533

getScanResult 561
getSelectedItemPosition 190
getSharedPreferences 254
getSimState 495
getSize 151
getString 110
getSupported* 477
getSystemService 324, 396, 409, 492, 555, 566, 587
getType 298, 443, 453
getUserData 508
getView 227
getWifiState 560
getWritableDatabase 284, 291
getX 641, 643, 646
getY 641, 643, 646
GIF 31, 103
glEsVersion 91
Gmail 27, 217, 313, 501
Google 15, 25, 34, 37, 46, 206, 235, 319, 420
    встроенные службы 29
    группы 38
Google Nexus One 14
Google Talk 27
GPS 14, 29, 64, 322, 325
GPS_PROVIDER 321, 325
GPX *см.* Формат для хранения и обмена данными GPS
GradientDrawable 160, 163
gradientRadius 163
GridLayoutAnimationController 614
GSM 27, 235, 492, 502

H

Handler 384, 389, 409
    postAtTime 390
    postDelayed 390
    потоки 390, 409
    таймеры 409
HCI *см.* Взаимодействие человека с компьютером
Hello World 52
    код 56
    компиляция и отладка 54
    разметка 104
heightMeasureSpec 150
HOME 204, 644
hookupButton 146
HTC 14
    Sense UI 27
HTML5 28
HTTP 463, 466,
HVGA 62, 91, 113, 169, 171, 175,

I

IM *см.* Система обмена мгновенными сообщениями
ImageButton 421
ImageView 166, 421, 428
IMEI 492
import 18, 597
in 102
includeInGlobalSearch 451
inflate 107, 145
initCompassView 155
initialLayout 422
innerRadius 161
innerRadiusRatio 162
InputStream 276, 302, 544
insert 287, 297, 300
instrumentation 94
Intent Filters *см.* Фильтры намерений
Intent Resolution *см.* Намерения, Утверждение
Intent *см.* Намерение
Intent.ACTION_CALL 492
Intent.ACTION_CALL_BUTTON 491
Intent.ACTION_DIAL 491
Intent.ACTION_VIEW 491
Intent.EXTRA_STREAM 501
intent-filter 203, 214, 160
IntentFilter 222

IntentReceiver 593
INTERNET 234, 326, 340
interpolator 105
INTERVAL_DAY 412
INTERVAL_FIFTEEN_MINUTES 411
INTERVAL_HALF_DAY 412
INTERVAL_HALF_HOUR 411
INTERVAL_HOUR 411
IPC 28, 589, 597, 601
iPhone *см.* Apple iPhone
IP-телефония 490, 492, 500, 561,
isAltPressed 644
isLooping 467
isModifierKey 645
isRouteDisplayed 340, 344
isShiftPressed 645
isSymPressed 645
isWifiEnabled 560
item 102, 186
item.getitemId 182
ItemClickListener 248
ItemClickListener 248
ItemizedOverlay 356, 358
itemizedOverlay<OverlayItem> 356
ItemizedOverlays 338

J

Java 13, 15, 23, 40, 44, 79, 195, 220, 339, 544, 600
  ME 24
  VM 37
  мидлеты 23
  типы 597
Java Development Kit 17, 46
java.io.File 276
java.lang.Thread 388
java.net.* 235
java.util.TimeZone 224
JDK *см.* Java Development Kit
JPEG 103, 476, 479
JPG 31
JRE *см.* Исполняющая среда Java

K

keyboard 115
keyboardHidden 113
keyCode 645
KeyListener 71
KML *см.* Язык разметки Keyhole

L

L2CAP *см.* Logical Link Control and Adaptation Protocol
label 422
LANGUAGE_MODEL_FREE_FORM 486
LANGUAGE_MODEL_WEB_SEARCH 486
largeScreens 91
LAUNCHER 121, 204
LayerDrawable 166, 167
LayoutAnimation 613, 614, 615
LayoutAnimationController 614
LayoutInflate 145
layoutOpt 79
LayoutParameters 137
LayoutParams 359
LBS *см.* Геолокационные сервисы
ledARGB 406
LENGTH_LONG 391
LENGTH_SHORT 391
LevelListDrawable 103, 167, 168, 173
libc 39, 40
LightingColorFilter 622
linear 163
LinearGradient 619
LinearLayout 75, 104, 132, 135, 136, 137, 138, 421, 434
Linkify 194, 216, 217, 218, 219, 250, 251
  типы ссылок 217
  шаблоны 218
Linkify.addLinks 217, 218

Linux Phone Standards Forum 24
LiPS *см.* Linux Phone Standards Forum
List 351, 597
ListActivity 128, 453
listen 497
listenUsingRfcommWithServiceRecord 529, 538
ListPreference 266
ListView 71, 128, 134
ListView 71, 75, 76, 103, 128, 132, 134, 139, 141, 167, 188, 189, 190, 208, 209, 210, 225, 226, 227, 228, 232, 241, 242, 243, 247, 248, 251, 295, 308, 310, 378, 380, 447, 454, 509, 511, 514, 545, 549, 551
Live Folders *см.* Живые каталоги
Live Wallpaper *см.* Живые обои
LiveFolders._ID 441
LiveFolders.Image 441
LiveFolders.NAME 441
locale 115
Locale 336, 487
LOCATION_SERVICE 324
Location-Based Services *см.* Геолокационные сервисы
LocationListener 328, 329, 330
LocationManager 320, 322, 324, 326, 328, 332, 369, 515
LocationManager.GPS_PROVIDER 322
LocationManager.NETWORK_PROVIDER 322
LocationProvider 320, 321, 322, 323, 324, 325, 328, 329, 330
Logical Link Control and Adaptation Protocol 537

M

Mac OS X 46
MAIN 121
main.xml 56
makeText 391
Manager.KEY_PROXIMITY_ENTERING 332
Map 597
MapActivity 128, 319, 338, 340, 341, 342, 344, 359, 360, 378
MapController 338, 343, 346, 356
MapView 128, 319, 336, 338, 339, 340, 342, 634
  MapController 343, 356
  добавление объекта 358
  ключ для API 339
  настройка элементов 342
Maps Google 27, 29, 35, 65, 203, 319, 335, 340, 342, 343
MapView.LayoutParams 359
MaskFilter 619, 621, 622,
MatchFilter 218
maxSDKVersion 88, 89
MCC *см.* Мобильный код страны
MD5 339, 340
MeasureSpec 151
Media Player *см.* Медиапроигрыватель
MediaPlayer 462
MediaRecorder 459, 468, 469, 470, 471, 472, 488
MediaScannerConnection 481
MediaScannerConnectionClient 481
MediaStore 311, 312, 394, 459, 460, 468, 469, 473, 481, 482, 488
  доступ 312
  добавление мультимедийных данных 481
MEID 492
MenuInflator 107
MessagePoster 552, 553,
middleColor 163
minHeight 422
minSDKVersion 88, 89
minWidth 422
MIRROR 621
MkSDCard 79
mm 102
MMS 468, 489, 500, 501, 527

MNC *см.* Код, мобильной сети
MotionEvent 640, 641, 642, 643, 645
MotoBlur 27
Motorola 14, 27, 32, 169
movePosition 612, 613
moveTo<местоположение> 287
moveToFirst 281
moveToNext 281
moveToPosition 281
moveToPrevious 281
MY_ACTION 373
MyLocationOverlay 338, 355, 356

N

Native Development Kit 40, 41
NDK *см.* Native Development Kit
NETWORK_STATE_CHANGED_ACTION 560
networkId 563
NetworkInfo 557, 559, 560
newWakeLock 593
NinePatch 103, 168, 173, 421, 626
normalScreens 91
NotificationManager 221, 367, 369, 394, 396, 400, 401, 402, 415
notify 400
notifyDataSetChanged 293
NPE *см.* NullPointerException
null 200, 227, 324, 334, 371, 474
NullPointerException 421
number 397

O

OHA *см.* Open Handset Alliance
OMA *см.* Open Mobile Alliance
onAccuracyChanged 568
onActivityResult 200, 261, 262, 473, 474, 487, 531, 535, 595
onAnimationEnd 607
onAnimationRepeat 607
onAnimationStart 607
onBind 369, 381, 602
onCallStateChanged 497
onCellLocation 497
onConfigurationChanged 115, 116, 118
onContextItemSelected 185, 190, 191
onCreate 56, 75, 76, 98, 103, 117, 120, 126, 127, 133, 144, 159, 188, 189, 192, 199, 207, 209, 210, 211, 229, 232, 243, 246, 248, 253, 257, 258, 259, 263, 268, 270, 272, 274, 284, 285, 290, 292, 294, 295, 303, 305, 310, 326, 330, 340, 359, 369, 377
onCreateContextMenu 184, 215
onCreateDialog 239, 249
onCreateOptionsMenu 107, 179, 182, 187, 248, 260
onDataActivity 499
onDataConnectionStateChanged 499
onDestroy 98, 126, 295
onDraw 92 100–101 108 471
OnInitListener 481
onItemClickListener 210, 550
onItemClickListener 210, 550
onKey 645
onKeyDown 140, 613, 639, 644
OnKeyListener 294, 645
onKeyListener 42 135
onKeyListener 76, 192, 229,
onKeyUp 153, 639, 644
onLocationChanged 328
onLowMemory 118
onMeasure 148–151, 155
OnMenuItemClickListener 181, 185
onOffsetsChanged 457
onOptionsItemSelected 179, 182, 190, 248, 260
onPause 127, 274, 570
onPostExecute 386, 415
onPrepareDialog 239, 249
onPrepareOptionsMenu 182, 189
onPreviewFrame 479

onProgressUpdate 386, 401
onReceive 68, 98, 221, 379, 385, 413, 423 439, 506, 593
onRestart 127
onRestore 359
onRestoreInstanceState 126, 253, 272
onResume 127, 364, 570, 586
onSaveInstanceState 126, 253, 272
onSensorChanged 568, 574, 580, 585
onServiceConnected 382
onServiceDisconnected 382
onServiceStateChanged 498
OnSharedPreferenceChangeListener 264, 268
onSharedPreferenceChangeListener 268
onStart 98, 126, 370, 373
onStartCommand 370
onStartCommand 370
onStop 126, 586
onSurfaceCreated 457
onTap 348, 350, 635
onTerminate 117
onTouchEvent 153, 457, 639
OnTouchListener 644
onTrackballEvent 153, 639, 645
onUpdate 423
onUpgrade 284
Open Handset Alliance 32, 34,
Open Mobile Alliance 24
openFileInput 275
openFileOutput 275
OpenGL 25, 28, 31, 39, 91, 147, 603, 636, 638
openOrCreateDatabase 285
openRawResource 276
orientation 115
OutputStream 544
oval 161
Overlay *см.* Наложение
OverlayItem 356
OverlayItems 338

P

p2p 19, 28, 528
package 88
Paint 143, 147, 149, 157, 349, 617–646
Palm Pre 23, 571
Parcelable 536, 589, 597–599
PARTIAL_WAKE_LOCK 594
PathDashPathEffect 623
pause 428, 463, 466
Pending Intents *см.* Намерения, Ожидающие
PendingIntent 225, 332, 413, 516
permission 93, 592
PHONE_NUMBERS 217
PhoneStateListener 496
PhoneStateListener.LISTEN_NONE 497
PictureCallback 479
pivotX 105, 165
pivotY 105, 165
PixelXorXfermode 624
play 428, 485
PNG 31, 103, 111, 168, 421
Point 348
populate 356
populateSpinners 257, 270
PorterDuffColorFilter 622
PorterDuffXfermode 624
post 390
postAtTime 390
postDelayed 390
postInvalidate 351
PowerManager 593
Preference Category *см.* Категории настроек
Preference Screens *см.* Экраны настроек
PreferenceActivity 264
PreferenceCategory 264
Preferences 255, 260
prepare 462, 466, 471
PreviewCallback 479
priority 563
ProgressBar 239, 421, 427

ProgressDialog 239
provider 93
pt 102
px 102

Q

Qualcomm 32
query 286, 297, 303
Quick Search Box *см.* Виджет быстрого поиска
QuickContactBadge 135
QVGA 14, 62, 91, 113, 169, 175,

R

R.drawable 108
R.string 108
radial 163
RadialGradient 619
RadioButton 134, 176, 178, 180, 186, 238
radius 161
RawContacts 313, 317
read 483
READ_CONTACTS 212, 313
READ_PHONE_STATE 492, 495
readPermission 592
RECEIVE_BOOT_COMPLETED 224
RECEIVE_SMS 506
receiver 93
RecognizerIntent 486
RECORD_AUDIO 469, 483
RECORD_VIDEO 469
rectangle 161
ReentrantLock 511
registerForContextMenu 184
registerReceiver 222
RelativeLayout 136, 172, 421
release 461, 470, 594
remapCoordinateSystem 581
RemoteViews 397
remove 111
removeItem 294
removeNetwork 563
removeUpdates 329
removeView 360
REPEAT 621
reqFiveWayNav 89
reqHardKeyboard 89
reqKeyboardType 89
reqNavigation 90
reqTouchScreen 90
requery 293,
requestLocationUpdates 328
requestReceived 513, 526
requires-permission 372
res 56, 112
res/ 99
res/anim 104, 614
res/drawable 103, 106, 429, 616
res/drawable-hdpi 174
res/drawable-ldpi 174
res/drawable-mdpi 174
res/layout 75, 104, 144
res/layout-large 174
res/layout-normal 174
res/layout-small 174
res/raw 276, 461
res/values 100, 111, 142, 451
res/xml 269, 422, 435, 447, 452, 455
Resource 276
resource 447
Resources 108
respond 512, 515, 516
RESTART 607
restoreUIState 274
RESULT_CANCELED 200
REVERSE 607
RFCOMM 537
ring 161
RingtonePreference 267
rotate 104, 105

RotateDrawable 165
RS232 537
RSSI_CHANGED_ACTION 561
RTC 339
RTC_WAKEUP 321
RTSP 410, 431
Runnable 553
runOnUiThread 389

S

S60 24
savePreferences 259, 523
SAX 235, 244
scale 104
ScaleDrawable 165
scaleHeight 165
scaleWidth 165
SCAN_MODE_CONNECTABLE 533
SCAN_MODE_CONNECTABLE_DISCOVERABLE 533
SCAN_MODE_NONE 534
SCAN_RESULTS_AVAILABLE_ACTION 561
scanFile 481
SCENE_MODE_* 475
scheduleLayoutAnimation 615
SCREEN_BRIGHT_WAKE_LOCK 594
SCREEN_DIM_WAKE_LOCK 594
SDK 14, 41, 44, 51, 63, 72, 78, 88, 131, 135, 154, 340, 507, 619
- менеджер 80
- установка 46

SearchManager 202
searchSettingsDescription 451, 452
SecurityException 373
seekTo 463, 466
SELECTED_ALTERNATIVE 204, 214
SelectionStateDrawable 421
SEND_SMS 502
SEND_TO 500
sendBroadcast 219, 223
sendDataMessage 505
sendIntent 593
sendMultipartTextMessage 504
sendOrderedBroadcast 223
sendTextMessage 502, 503
Sensor.TYPE_ACCELEROMETER 566
Sensor.TYPE_GYROSCOPE 566
Sensor.TYPE_LIGHT 566
Sensor.TYPE_MAGNETIC_FIELD 567
Sensor.TYPE_ORIENTATION 567
Sensor.TYPE_PRESSSURE 567
Sensor.TYPE_PROXIMITY 567
Sensor.TYPE_TEMPERATURE 567
SensorEventListener 568–570, 574, 579
SensorManager 566, 569, 570, 573, 580
SensorManager.SENSOR_DELAY_GAME 569
SensorManager.SENSOR_DELAY_NORMAL 569
SensorManager.SENSOR_DELAY_UI 569
SensorManager.SENSOR_STATUS_ACCURACY_HIGH 569
SensorManager.SENSOR_STATUS_ACCURACY_LOW 569
SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM 569
SensorManager.SENSOR_STATUS_UNRELIABLE 569
SensorManger.SENSOR_DELAY_FASTEST 569
sentIntent 503
serviceBinder 382
ServiceConnection 382
Services *см.* Сервисы
ServiceState 498
set 105
set* 475
set[audio/video]Encoder 470
setAdapter 228
setAlarm 524
setAnimationListener 607
setAttribute 480

setAudioSource 470
setBackgroundDrawable 616
setBackgroundResource 616
setBuiltInZoomControls 343
setCenter 344
setCheckable 180
setChecked 180
setColor 618
setContentView 56, 103, 103, 120, 133, 237, 360
setDataSource 462, 465
setDisplay 464
setGravity 392
setHeaderIcon 183
setIcon 183
setImageLevel 168
setInexactRepeating 411
setJpegQuality 476
setJpegThumbnailQuality 476
setJpegThumbnailSize 476
setKeepScreenOn 463
setLatestEventInfo 384, 397
setLayoutParams 137
setLevel 166
setLooping 467
setMeasuredDimension 149, 151, 155
setNetworkPreference 558
setOnClickPendingIntent 428
setOnKeyListener 645
setOnTouchListener 644
setOutputFile 470
setOutputFormat 471
setParameters 475
setPictureFormat 476
setPictureSize 476
setPitch 595
setPreviewCallback 479
setPreviewDisplay 472, 478
setPreviewFormat 479
setPreviewFrameRate 476
setPreviewSize 378 476
setRepeatCount 607
setRepeating 411
setRepeatMode 607
setResult 199
setScreenOnWhilePlaying 467
setShader 620
setShortcut 181
setSpeechRate 595
setStyle 618
Settings 311
settingsActivity 455
setTitle 237
Setup Auto Responder 518
setupListenButton 548
setupListView 549
setupSearchButton 550
setVideoPath 463
setVideoSource 470
setVideoUri 463
setView 392
setViewVisibility 427
setVolume 467
setZoom 343
Shader 619, 625, 627, 630, 634
ShapeDrawable 160
SharedPreferences 253
SharedPreferences.Editor 254
SHOW_DAMAGE 205
showActivity 443
showDialog 240
ShutterCallback 479
SIM 112, 264, 492
    считывание данных 495
SimpleCursorAdapter 209, 232, 251
size 151, 356
smallScreens 91
SMS 489, 500
    передача с помощью Намерения 501
    отправка вручную 502

отслеживание доставки 503
проверка на соответствие максимально допустимой длине 504
отслеживание входящих 505
моделирование в эмуляторе 507
бинарные 508
SmsManager 489, 500, 502, 504
SmsManager.RESULT_ERROR_GENERIC_FAILURE 503
SmsManager.RESULT_ERROR_NULL_PDU 503
SmsManager.RESULT_ERROR_RADIO_OFF 503
SmsMessage 502, 506
Sony Ericsson 14, 27, 169
sound 405
sp 102
Spinner 134, 255
SQLiteDatabase 287, 304
SSID 561–563
SSL 39
start 466, 471, 616
START_FLAG_REDELIVERY 372
START_FLAG_Retry 372
START_NOT_STICKY 288
START_REDELIVER_INTENT 288
START_STICKY 414
startActivity 182, 196, 205, 217, 267, 501
startActivityForResult 196, 260, 468, 531, 594
startAnimation 607
startAutoResponder 525
startColor 163
startDiscovery 536, 541
startForeground 299
startManagingCursor 211 222
startNewActivityForResult 387
startNextMatchingActivity 147
startOffset 64 65
startRecording 384
startService 369–374
STATE_EMERGENCY_ONLY 498
STATE_IN_SERVICE 498
STATE_OFF 531
STATE_ON 530
STATE_OUT_OF_SERVICE 498
STATE_POWER_OFF 498
STATE_TURNING_OFF 531
STATE_TURNING_ON 530
StateListDrawable 167, 357
status 563
stop 466, 471, 595
stopForeground 384
stopManagingCursor 211
stopPlayback 281
stopSelf 370–372, 415
stopService 370–372
StreetView 27
String 157, 597
String.format 101
STROKE 618
STROKE_AND_FILL 618
SUBPIXEL_TEXT_FLAG 624
SUGGEST_COLUMN 449
SUGGEST_COLUMN_TEXT_1 449
SumPathEffect 623
SUPPLICANT_CONNECTION_CHANGE_ACTION 560
supports-screens 91
Surface 147, 455, 457
surfaceCreated 457, 465
SurfaceHolder 464, 478
SurfaceHolder.Callback 465, 636
SurfaceView 147, 464, 472, 478, 590, 635–646
sweep 163
SweepGradient 619
switchUI 548, 550, 552

T

TabActivity 128
Tablelayout 136
TAG_* 480

takePicture 479
takePicture 479
targetSDKVersion 88
tel: 202, 206, 490
TelephonyManager 492, 495,
TelephonyManager.CALL_STATE_IDLE 497
TelephonyManager.CALL_STATE_OFFHOOK 497
TelephonyManager.CALL_STATE_RINGING 497
text to speech 594
TextView 104, 120, 134
thickness 161
thicknessRatio 162
Thread 368, 384, 388, 636
TileMode 621
TimePickerDialog 239
timestamp 482, 506, 568
T-Mobile 14, 32
Toast 237, 368
  вывод 390
  настройка 392
toDegrees 105, 165,
toPixel 349
toString 134, 226, 242
touchscreen 115
Traceview 79
TransformFilter 218, 219
transformUrl 219
translate 104, 625
TTS *см.* text to speech
type 163, 501,
TYPE_ACCELEROMETER 566
TYPE_GYROSCOPE 566
TYPE_LIGHT 566
TYPE_MAGNETIC FIELD 567
TYPE_ORIENTATION 567
TYPE_PRESSURE 567
TYPE_PROXIMITY 567
TYPE_TEMPERATURE 567

U

UI. *см.* Пользовательский интерфейс
UIQ 24
unregisterReceiver 222
UNSPECIFIED 151
update 287, 297, 301
updateAppWidget 425
updateArray 293
updateFromPreferences 262
updateNetwork 563
updateOrientation 584
updatePeriodMillis 422
updateUIFromPreferences 258, 522
UriMatcher 296, 448
UserDictionary 311
uses-configuration 89
uses-feature 90
uses-library 340
uses-permission 93, 212, 224
uses-sdk 88
UUID *см.* Универсальный уникальный идентификатор
UX *см.* Впечатления от использования

V

value 448
values 56, 99, 573
versionCode 88
versionName 88
vibrate 405, 587
VIBRATE 587
Vibrator 587
VideoView 366–367
View Group *см.* Группы представлений
View *см.* Представления
ViewFlipper 135
ViewGroup 120, 131, 135, 145, 615
VM *см.* Виртуальная машина
VOIP *см.* IP-телефония

W

WakeLock 589, 593
WallpaperService 455
WallpaperService.Engine 456
WEB_URLS 217
WebKit 25, 26, 28, 39, 233, 235
Where Am I?, приложение:
- геокодирование 336
- добавление аннотаций 351
- добавление карты 344
- изменение местоположения в нем 329
- создание 325

WHITE_BALANCE_* 476
widthMeasureSpec 150
Wi-Fi 27, 29, 33, 234
- отслеживание соединения 560
- создание конфигураций 563
- точки доступа 561
- управление настройками 562

WIFI_STATE_CHANGED_ACTION 560
WifiConfiguration 562
WifiManager 555, 559, 562
Windows 17, 35, 44, 46, 83, 339, 617
Windows Mobile 23,
WQVGA432 175
wrap_content 137, 172
WRITE_CONTACTS 317
writePermission 592
WVGA 14, 62, 91, 113, 169

X

Xfermode 624
XML 48, 57, 88, 94, 99, 103, 120, 235
- Drawable 160
- разметка 136, 145, 160, 264

xmlns:android 88
Xperia X10 169

Y

Yahoo! Pipes 603
YouTube 27

Z

zoomIn 343
zoomOut 343

А

Адаптер 191, 225, 258, 281
- настройка 226, 228
- привязка данных 228
- применение 228
- создание 228
- стандартные 226

Акселерометр 28, 566, 571–581, 588, 613
Активность 41, 51–59, 85–129
- «Активность не отвечает» 68
- активное состояние 122, 127
- видимое состояние 122, 127
- жизненный цикл 121–128
- классы 128
- меню 179–187, 215
- неактивное состояние 79, 123
- неявный запуск 198, 206
- остановленное состояние 123
- полноценное состояние 122, 126
- приостановленное состояние 122
- создание 119–121
- сохранение и восстановление 272
- стеки (очереди) 121
- явный запуск 197

Активные процессы 97
Анимация 104
- атрибуты 105
- изменение пользовательского интерфейса 608
- мультипликационная 604
- на расчете промежуточных кадров 104, 605, 607
- повторяющаяся бесконечно 607
- пошаговая 106, 615
- простая 106
- разметки 613

Асинхронные задачи 385–387
Аудио 31
воспроизведение 463, 484
запись 468, 482
инициализация 462
несжатое 482
проигрывание 460
режим повторения 467
упаковка 461

Б

Базы данных 35, 278
Библиотеки 39, 42
в составе SDK 14
с открытым исходным кодом 25
мультимедийные 28
графические 29
комплексные 31
SQLite 278
картографическая 340
для преобразования текста в речь 589, 594
API 21, 28, 37, 45

В

Взаимодействие человека с компьютером 131
Вибрация 30, 394,405,
управление 587
Виджет быстрого поиска 446, 450
добавление поисковых возможностей в Earthquake 451
добавление поисковых возможностей в приложение 447
доставка результатов 450
Видео 22, 28, 39, 311
воспроизведение 463
запись 468, 470,
инициализация 465
предварительный просмотр 471
проигрывание 460
Виджеты 59, 87, 134, 418
IntentReceiver 419, 423
RemoteViews 419
XML 419
добавление интерактивности 427
минимальная частота обновления 433
создание 419
Виртуальная машина 23, 25, 31, 37, 40, 44, 54, 78, 96
Виртуальные устройства в Android 53, 78
ВМ *см.* Виртуальная машина
Впечатления от использования 131

Г

Геокодировщик 333
Геокодирование:
обратное 29, 320, 333, 334
прямое 333, 335
Геолокационные сервисы 65, 320, 508
выбор источника 322
изменение местоположения 321
поиск местоположения 324
тестирование 320
Гироскоп 566
Группы представлений 132, 613

Д

Диалог принудительного закрытия приложения 68, 334, 385
Диалоговые окна 236
-Активности 239
классы 237
отображение 240
специальные для ввода данных 239
управление 240
Дисплей 14, 21, 171, 319
высота 113
небольшой 23, 29, 36, 62, 139, 183
плотность пикселов 113, 170, 625

размер 170
соотношение сторон 170
типы 113
ширина 113
Дозвон 197, 491
Домашний экран 417

Ж

Живые каталоги 31, 36, 417, 439–446
finish 442
создание 440, 443
Живые обои 31, 36, 417 455–457
создание 455

З

Звонки:
вибро- 405
входящие 27, 66, 316
имитация 48
инициирование 489
история 311
исходящие 224
отслеживание 497
Значки 181

И

Инструмент для создания пакетов 78, 108
Исполняющая среда Java 46
Интернет 33, 63, 194, 234
безопасность 39,
-ресурсы 234
-сервисы 602
Источники данных 65, 86, 279, 592
URI 295
добавление 300
доступ к файлам 302
запрос 299
обновление 301
общие 279
поиск 322
предоставление доступа 297
создание 295, 303, 308
стандартные 310
удаление 301

К

Камера:
автофокусирвка 477
метаданные EXIF 480
настройки камеры 475
параметры 475
предварительный просмотр 460, 477
создание фотографий 479
Категории настроек 264
Клавиатура 80, 90, 104, 113, 639, 644
qwerty 90
подсветка 593
сокращенные команды 176, 181
тип 89, 113
Код:
запроса 198, 200
мобильной сети 112, 113, 493, 495
результирующий 199
Компас 16, 154, 565, 571
создание 582
усовершенствованный циферблат 626
Композитные ресурсы для рисования 165
Контекст 268, 462
Контурные эффекты 621, 623
Конфиденциальность отслеживание местоположения 329
Краткие заголовки 181
Курсор 202, 209, 227, 280, 287, 314
базы данных 280
возможности навигации 281
Кэш 64, 234

Л

Латентность 60, 63, 82, 500
Локализация 28

М

Магнитное поле 567, 570, 578
Манифест 48, 86, 88
 редактор 94
Мастер создания проектов для Android 94
Медиапроигрыватель 459, 460
Менеджер Активностей 41, 36
Менеджер виртуальных устройств 38, 78, 79
Менеджер компоновки *см.* Разметка
Менеджер ресурсов 42
Меню 176
 выбор пунктов 182
 динамическое изменение пунктов 182
 добавление в приложение To-Do List
 дочернее 178, 183
 контекстное 184
 описание для Активности 179
 описание с помощью XML 185
 параметры пунктов 180
 расширенное 177
 со значками 177
Миниатюры 476
Мобильный код страны 112, 113, 493, 495
Мультимедиа 30, 39, 311

Н

Наложение 338
 Детализированное 356
 добавление 350
 рисование 349
 создание 347
 удаление 350
 улучшение 634
Намерения 87, 181, 195–198
 «липкие» 223
 для запуска Активностей 201
 неявные 197, 199, 203
 обслуживание 203
 Ожидающие 225
 трансляция событий 219
 упорядоченные 223
 Утверждение 196, 204
 явные 196
Независимость от плотности пикселов 179

О

Облачные вычисления 602
Оповещения о близости нахождения 320

П

Панель быстрого поиска 36
Переключатели *см.* RadioButton
Подсветка 394, 463, 467, 593
Полномочия 591
Полупрозрачность 603, 605, 617, 619
Пользовательский интерфейс 127, 131, 252, 603
Поток 126
 GUI 386
 графический 247, 385, 389, 625
 дочерний 388
 из Интернета 234, 460
 рабочий 82, 393
 создание 388
 файловый 275
 фоновый 147, 384, 261, 334, 367, 381, 384, 464, 552
Представления 42, 56, 86, 120, 131, 132
 дочерние 132
 создание пользовательского интерфейса 133
 виды 134
 создание 138
 изменение 139
 составные элементы управления 144
 нестандартные 147, 159

интерактивность 153
привязка к Адаптеру 225
добавление на карту 358
Приемники широковещательных намерений 194, 384, 419, 506
обеспечение полномочий 592
отслеживание трансляций 220
регистрация 222
Принадлежность данных 206
Приоритет приложений 97, 121, 383
Процессы:
активные 97
видимые 97, 98
с запущенными Сервисами 97, 98
фоновые 97, 98
холостые 97, 98

Р

Разметка 99, 103, 135, 172, 263, 340, 392
вложенная 138
классы 135
нестандартная 398
оптимизация 138
создание 136
Разрешение 31, 60, 78, 80, 86, 91, 102, 109, 130, 137, 160, 163, 169–176, 468, 473
Распознавание речи 486
Растровые изображения 103, 147, 173, 426, 476, 616, 625
Расширяемость 214, 317
Регулярное выражение 217–219
Рубин Энди 25

С

Светодиоды 66, 82, 87, 396
индикация 406
Сглаживание 617, 624
Сенсорный экран *см.* Дисплей
Сервис ConnectivityManager 555, 556, 557, 558, 559
Сервисы 59, 86, 98, 368–384
запуск 373
привязка Активностей 381
создание 369, 433
Сигнализация 368, 408, 431
повторяющаяся 411
создание 410
Система обмена мгновенными сообщениями 70, 135
Система управления окнами 68
Служба коротких сообщений *см.* SMS
Служба мультимедийных сообщений *см.* MMS
Статусная строка 66
разметка 384
расширенная 395, 400

Т

Темы 71, 102, 110, 120
Трафик 71, 127, 557
Трекболы 89, 153, 174, 184, 571, 639, 645

У

Уведомления 30, 87, 394–408, 539
настойчивые 408
показ 400
создание 396
текущие 407
Универсальный уникальный идентификатор 538, 542

Ф

Фильтры намерений 88, 96, 121, 185, 203, 208
для создания дополнений 214
принцип работы 205
Флажки *см.* CheckBox
Флеш:
память 31
сообщения 30
Формат для хранения и обмена данными GPS 321
Фреймворк 40, 160, 169, 263

Х

Холст 348, 616

Ц

Цветовые фильтры 590, 622

Ш

Шейдеры 590, 603, 618–621
Широковещательные приемники 59, 68, 87, 98, 195
 регистрация 93, 127, 222
 создание 220

Э

Экран *см.* Дисплей
Экраны настроек 264
 импорт системных 265
 поиск 268
Эмулятор Android *см.* Эмулятор
Эмулятор 38, 44, 55, 63, 78, 82
 и Bluetooth 545
 и вибрация 420
 и светодиод 421
 моделирование SMS 507
 настройка 320
 оболочки 174

Я

Ядро Linux 24, 37, 38, 39, 590
Язык и регион 113
Язык описания интерфейсов в Android 382, 597–602
 реализация интерфейсов 597, 600
Язык разметки Keyhole 321, 322

Производственно-практическое издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Рето Майер

ANDROID 2
Программирование приложений
для планшетных компьютеров и смартфонов

Ответственный редактор *В. Обручев*
Художественный редактор *Н. Биржаков*

ООО «Издательство «Эксмо»
127299, Москва, ул. Клары Цеткин, д. 18/5. Тел. 411-68-86, 956-39-21.
Home page: **www.eksmo.ru** E-mail: **info@eksmo.ru**

Подписано в печать 22.06.2011. Формат 70x100 $^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 54,44.
Тираж экз. Заказ

ISBN 978-5-699-50323-0

9 785699 503230 >

Книга представляет собой комплексное руководство для программистов, желающих научиться создавать приложения для мобильной платформы Android.

Это практический курс по написанию приложений на базе второй версии Android SDK. Все теоретические сведения закрепляются максимально приближенными к реальным задачам примерами. Изложение материала предполагает, что читатель владеет основами программирования и базовым уровнем языка Java (второе желательно, но не обязательно).

Информация будет полезной как для опытных разработчиков (они могут использовать ее как справочник, пропустив первые, элементарные главы), так и для тех, кто делает свои первые шаги в сфере написания мобильных приложений для Android.

Эта книга:

- Описывает Android как платформу разработки и показывает, как создавать на ней мобильные приложения максимально эффективно
- На профессиональном уровне рассматривает готовые компоненты для приложений Android
- Уделяет большое внимание шаблонам и другим технологиям, необходимым для создания полнофункциональных, не зависящих от разрешения экрана пользовательских интерфейсов
- Разъясняет, как обобщать и сохранять данные
- Обучает техникам для создания приложений с использованием карт, а также использующих навигационные сервисы вроде GPS
- Объясняет, как создавать фоновые сервисы, напоминания и сообщения
- Демонстрирует создание интерактивных приложений
- Дает обзор API для работы с телефонией, сетью и Bluetooth
- Описывает использование аппаратного обеспечения.

РЕТО МАЙЕР родом из города Перт в Австралии, но сейчас проживает в Лондоне.

В данный момент Рето работает в компании Google (создатель Android), помогая разработчикам создавать самые лучшие приложения для платформы Android. Рето — опытный программист, за плечами которого более 10 лет проектирования приложений с графическим интерфейсом. Прежде чем перейти в Google, он разрабатывал программные решения в самых разных областях, включая финансы и нефтегазовые проекты. Всегда находясь на переднем крае прогресса новых технологий, Рето принимал участие в проекте Android с момента его первого выпуска в 2007 году.

ЭКСМО


ISBN 978-5-699-50323-0
9 785699 503230 >